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.iit.ac.in

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

ABSTRACT
Frequent patterns are an important class of regularities that exist in a transaction database. Certain frequent patterns with low minimum support (\(\text{minsup}\)) value can provide useful information in many real-world applications. However, extraction of these frequent patterns with single \(\text{minsup}\)-based frequent pattern mining algorithms such as Apriori and FP-growth leads to “rare item problem.” That is, at high \(\text{minsup}\) value, the frequent patterns with low \(\text{minsup}\) are missed, and at low \(\text{minsup}\) value, the number of frequent patterns explodes. In the literature, “multiple \(\text{minsup}\) 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.

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 (\(\text{minsup}\)) framework” to discover complete set of frequent patterns. \(\text{Minsup}\) 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 \(\text{minsup}\)-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 \(\text{minsup}\) framework” leads to the rare item problem which is as follows: At high \(\text{minsup}\), the frequent patterns containing rare items will be missed, and at low \(\text{minsup}\), 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 \(\text{minsup}\) framework.” In this framework, each pattern can satisfy a different \(\text{minsup}\) depending upon the items within it. The frequent patterns discovered through “multiple \(\text{minsup}\) framework” do not satisfy downward closure property. As a result, this property cannot be used for minimizing the search space in multiple \(\text{minsup}\)-based frequent pattern mining algorithms.

In the literature, an Apriori-like algorithm known as Multiple Support Apriori (MS/Apriori) was proposed to find frequent patterns with “multiple \(\text{minsup}\) 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 \textit{minsups} framework,” the CFP-
growth algorithm has to carry out exhaustive search in the
constructed \textit{Tree} structure. In this paper we propose an im-
proved 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 \textit{rare item problem} with the usage of
traditional data mining techniques to discover knowledge
involving rare items was introduced in [13]. In [10], “mul-
tiple \textit{minsups framework}” has been introduced to address
\textit{rare item problem}, and MSApriori algorithm was proposed
for extracting frequent patterns. An \textit{FP-growth-like algo-
rithm} [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 \textit{relative support}
has been introduced, and an Apriori-like algorithm has been
proposed for mining frequent patterns containing both fre-
quent and rare items. An Apriori-like approach which tries
to use a different \textit{minsup} at each level of iteration has been
discussed in [11]. A stochastic mixture model based on neg-
ative 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 \textit{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 spe-
cify items’ \textit{MIS} values. In [9], a new interestingness measure,
called \textit{item-to-pattern difference}, has been used along with
the “multiple \textit{minsups framework}” to discover frequent pat-
terns in the databases, where frequencies of the items’ vary
widely. An effort has been made to extend the notion of mul-
tiple 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 \textit{tree} struc-
ture. It is to be noted that the algorithm discussed in [7]
does not satisfy \textit{downward closure property}.

In this paper, we investigated approaches to reduce the
search space while extracting frequent patterns and pro-
posed two additional pruning techniques which significantly
reduces the search space by avoiding exhaustive search while
extracting frequent patterns from a \textit{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 Sec-
tion 3, we discuss the CFP-growth algorithm and its per-
formance 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 pat-
terns, \textit{rare item problem} and the extended model of frequent
patterns based on multiple \textit{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 transac-
tion database \( DB = \{T_1, T_2, \cdots, T_m\} \), 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 \textit{support} of a pattern (or an itemset) \( X \), denoted as \( S(X) \),
is the number transactions containing \( X \) in \( DB \). The pat-
tern \( X \) can be \textit{frequent} if its support is no less than a user-
defined minimum support (\textit{minsup}) threshold value, i.e.,
\( S(X) \geq \text{\textit{minsup}} \). A pattern containing \( k \) number of items
is a \( k \)-pattern. The support of a pattern can also be rep-
resented in percentage of \( |DB| \). In this paper, we use the
terms “itemset” and “pattern” interchangeably.

\textbf{Example 2:} Consider the transaction database of 20
transactions shown in Table 1. The set of items \( I = \{a, b, c, d, e, f, g, h\} \)
be a set of items, and a transac-
tion database \( DB = \{T_1, T_2, \cdots, T_m\} \), 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 \textit{support} of a pattern (or an itemset) \( X \), denoted as \( S(X) \),
is the number transactions containing \( X \) in \( DB \). The pat-
tern \( X \) can be \textit{frequent} if its support is no less than a user-
defined minimum support (\textit{minsup}) threshold value, i.e.,
\( S(X) \geq \text{\textit{minsup}} \). A pattern containing \( k \) number of items
is a \( k \)-pattern. The support of a pattern can also be rep-
resented in percentage of \( |DB| \). In this paper, we use the
terms “itemset” and “pattern” interchangeably.

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a, b</td>
<td>11</td>
<td>a, b</td>
</tr>
<tr>
<td>2</td>
<td>a, c, f</td>
<td>12</td>
<td>a, c</td>
</tr>
<tr>
<td>3</td>
<td>c, d</td>
<td>13</td>
<td>a, b</td>
</tr>
<tr>
<td>4</td>
<td>a, b, h</td>
<td>14</td>
<td>b, e, f</td>
</tr>
<tr>
<td>5</td>
<td>c, d</td>
<td>15</td>
<td>c, d</td>
</tr>
<tr>
<td>6</td>
<td>a, c</td>
<td>16</td>
<td>a, b, d</td>
</tr>
<tr>
<td>7</td>
<td>a, b</td>
<td>17</td>
<td>c, d</td>
</tr>
<tr>
<td>8</td>
<td>e, f</td>
<td>18</td>
<td>a, c</td>
</tr>
<tr>
<td>9</td>
<td>c, d, g</td>
<td>19</td>
<td>a, b, e</td>
</tr>
<tr>
<td>10</td>
<td>a, b</td>
<td>20</td>
<td>c, d</td>
</tr>
</tbody>
</table>

Apriori [1] and FP-growth [5] are the two popular algo-
rithms to mine frequent patterns. Apriori uses candidate-
generate-and-test-approach to discover the complete set of
frequent patterns. FP-growth employs pattern-growth tech-
nique to discover complete set of frequent patterns. In the
literature, it has been shown that FP-growth performs bet-
ter 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 pat-
terns containing rare items.

i. If \textit{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 \textit{minsup}. However,
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 minsup framework.” The terms “T” and “F” respectively denote the frequent patterns generated and have not generated in single and multiple minsup frameworks.

<table>
<thead>
<tr>
<th>Patterns</th>
<th>S</th>
<th>MIS</th>
<th>SMF</th>
<th>MMF</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>12</td>
<td>10</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>c</td>
<td>9</td>
<td>10</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>b</td>
<td>9</td>
<td>8</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>d</td>
<td>7</td>
<td>6</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>e</td>
<td>4</td>
<td>3</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>f</td>
<td>3</td>
<td>3</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>ab</td>
<td>8</td>
<td>-</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>ac</td>
<td>3</td>
<td>-</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>ef</td>
<td>3</td>
<td>-</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

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 minsup [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).

\[ \text{minsup}(X) = \min \left\{ \text{MIS}(i_1), \text{MIS}(i_2), \ldots, \text{MIS}(i_k) \right\} \] (1)

where, \( X = \{i_1, i_2, \ldots, i_k\}, 1 \leq k \leq n \), 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 \( S(ac) < \text{minsup} = \min(\text{MIS}(a), \text{MIS}(c)) \).

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.

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.
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’, ‘c’ 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 straightforward 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 minsup” 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, \(I = \{i_1, i_2, \ldots, i_n\}\), such that \(MIS(i_1) \geq MIS(i_2) \geq \cdots \geq MIS(i_n)\). Let \(X = \{i_j, \ldots, i_k\} \subseteq I\), where \(1 \leq j \leq k \leq n\), be a pattern. If \(X\) is frequent, then \(S(X) \geq \min\{MIS(i_j), \ldots, MIS(i_k)\}\). 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, \(I = \{i_1, i_2, \ldots, i_n\}\). Let \(MIS(i_k)\), where \(i_k \in I\), be the user-specified MIS values such that \(MIS(i_1) \geq MIS(i_2) \geq \cdots \geq MIS(i_n)\). Let \(A \subseteq X\) 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 minsup 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 straightforward 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 \(X = \{i_1, i_2, \ldots, i_k\} \subseteq I\), where \(1 \leq k \leq n\), is a pattern such that \(S(X) < \text{LMS}\), then \(S(X) < \min\{MIS(i_1), MIS(i_2), \ldots, MIS(i_k)\}\).

Property 3. If \(X\) and \(Y\) are two patterns such that \(X \subset Y\) and \(S(X) < \text{LMS}\), then \(S(Y) < \text{LMS}\).

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 \(i\) 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 \(i\) will be the lowest MIS value among all the items in the \(conditional\ pattern\ base\). From the definition of \(minsupt\) in multiple \(minsupt\) framework, it turns out that any frequent pattern that is going to be generated from the \(conditional\ pattern\ base\) of \(i\) should satisfy MIS value of \(i\). Thus, we call the MIS value of the suffix item \(i\) as the \(conditional\ minsupt\). The correctness of this idea is shown in Lemma 4.3.

**Lemma 4.3.** Let \(\alpha\) be a pattern in MIS-tree and \(S_{\alpha}\) be the support of \(\alpha\). Also, let \(minsupt_{\alpha}\) be the \(minsupt\) that \(\alpha\) has to satisfy, \(B\) be \(\alpha\)'s \(conditional\ pattern\ base\), and \(b\) be an item in \(B\). The support of \(\beta\) in \(B\) is \(S_{B}(\beta)\) and \(MIS_{\beta}\) be the \(\beta\)'s MIS value. The \(minsupt\) of pattern \((\alpha, \beta)\) is \(minsupt_{\alpha}\).

**Proof.** According to the definition of MIS-tree, \(MIS_{B}(\beta)\) will always be greater than or equal to the \(minsupt_{\alpha}\). Therefore, \(minsupt\) of \((\alpha, \beta)\) is \(minsupt_{\alpha}\).

**Example 10:** Consider the compact MIS-tree shown in Figure 2(b). For the (suffix) item \(f\), the \(conditional\ prefix\ paths\) are \((a, c: 1), (c: 1)\) and \((b, c: 1)\). The item having lowest MIS value among all the items ‘\(a\)’, ‘\(b\)’, ‘\(c\)’ 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 \(minsupt = MIS(f)\). Thus, \(MIS(f)\) is considered as \(conditional\ minsupt\) 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 \(\alpha\) be a pattern in MIS-tree and \(S_{\alpha}\) be the support of \(\alpha\). Also, let \(minsupt_{\alpha}\) be the \(minsupt\) that \(\alpha\) has to satisfy, \(B\) be \(\alpha\)'s \(conditional\ pattern\ base\), and \(b\) be an item in \(B\). The support of \(\beta\) in \(B\) is \(S_{B}(\beta)\) and \(MIS_{\beta}\) be the \(\beta\)'s MIS value. If \(\alpha\) is infrequent, then the pattern \((\alpha, \beta)\) 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 \(\alpha\) in the transaction database. If an item \(\beta\) appears in \(B\) for \(n\) times, it appears with \(\alpha\) in \(n\) times. From the definition of frequent pattern used in the minimum constraint model, the \(minsupt\) of \((\alpha, \beta)\) is \(minimum(\implies minsupt_{\alpha}, \implies MIS_{\beta}) = minsupt_{\alpha}\). As \(S_{\alpha} \leq minsupt_{\alpha}\), the \(S_{(\alpha, \beta)} \leq minsupt_{\alpha}\) (apriori property [1]). Therefore, \((\alpha, \beta)\) 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 straightforward to prove from the “conditional minsupt” 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.](image-url)
Algorithm 1 MIS-tree (DB: transaction database, I: item-set 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: \( \) end if
14: \( \) end for
15: \( \) Name the resulting table as MinFrequentItemHeaderTable.
16: \( \) Call InfrequentLeafNodePruning(Tree).
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 InfrequentMerge(Tree).
27: \( \) Call InfrequentLeafNodePruning(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 \( i \) 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

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. 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. 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 ‘c’, ‘d’ and ‘f’ in \( L \) order will result in a branch where ‘c’ is linked to root, ‘d’ 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, \( \langle c: 1 \rangle \) and \( \langle f: 1 \rangle \), where ‘c’ is linked to ‘a’ and ‘f’ is linked to ‘e’. 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.

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

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
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 (T104D100K) and real-world datasets (BMS-WebView-I [15], Mushroom and Kosarak). T104D100K, BMS-WebView-I 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.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Transactions</th>
<th>Distinct Items</th>
<th>Max. Trans. Size</th>
<th>Avg. Trans. Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>T104D100K</td>
<td>100000</td>
<td>870</td>
<td>29</td>
<td>10.102</td>
</tr>
<tr>
<td>BMS-WebView-I</td>
<td>59602</td>
<td>497</td>
<td>267</td>
<td>2.5</td>
</tr>
<tr>
<td>Mushroom</td>
<td>8124</td>
<td>119</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>Kosarak</td>
<td>999002</td>
<td>41270</td>
<td>2498</td>
<td>8.1</td>
</tr>
</tbody>
</table>

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

\[ M(i_j) = \max(\beta \times f(i_j), LS) \]  

The \( f(i_j) \) and \( M(i_j) \) 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 \( \text{minsup} \) in traditional frequent pattern mining. If \( \beta = 1 \) and \( f(i_j) \geq LS \), then \( M(i_j) = f(i_j) \).

5.1 Experiment 1

In this experiment, both \( LS \) and \( \text{minsup} \) values are set at 0.1% for T104D100K and BMS-WebView-I datasets. For the Mushroom dataset, both \( LS \) and \( \text{minsup} \) values are set at 10% as it is a dense dataset. To show how \( \beta \) affects the number of frequent patterns found and the performance of the algorithms, we fixed \( \beta = \frac{1}{2} \) and varied \( \alpha \). In the sparse datasets (T104D100K and BMS-WebView-I), \( \alpha \) is varied from 1 to 20. In the dense dataset (Mushroom), \( \alpha \) 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 \( \alpha \) becomes larger, the number of frequent patterns found by the method gets closer to the number of frequent patterns found with the single \( \text{minsup} \) framework. The reason is as follows. At higher values of \( \alpha \), the items’ MIS values become equals to \( LS \). As a result, the performance of the “multiple \( \text{minsup} \) framework” is same as the “single \( \text{minsup} \) framework”. It can also be observed that the above phenomenon happens at higher values of \( \alpha \) in the sparse datasets (Figure 3(a) and 3(b)) and at lower values of \( \alpha \) in the dense dataset (Figure 3(c)).
Frequent patterns generated when minsup = 1%, the frequent patterns generated when minsup = 25% and minsup = 4 were interesting to the users. It can be observed that, at α = 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 β = 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...
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 infrequent 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


