# Efficient Discovery of Correlated Patterns in Transactional Databases using Items' Support Intervals

R. Uday kiran and Masaru Kitsuregawa

Institute of Industrial Science, The University of Tokyo, Komaba-ku, Tokyo, Japan.
{uday_rage,kitsure}@tkl.iis.u-tokyo.ac.jp
http://www.researchweb.iiit.ac.in/~uday_rage
http://www.tkl.iis.u-tokyo.ac.jp/Kilab/Members/memo/kitsure_e.html

**Abstract.** Correlated patterns are an important class of regularities that exist in a transactional database. CoMine uses pattern-growth technique to discover the complete set of correlated patterns that satisfy the user-defined minimum support and minimum all-confidence constraints. The technique involves compacting the database into FP-tree, and mining it recursively by building conditional pattern bases (CPB) for each item (or suffix pattern) in FP-tree. The CPB of the suffix pattern in CoMine represents the set of complete prefix paths in FP-tree co-occurring with itself. Thus, CoMine implicitly assumes that the suffix pattern can concatenate with all items in its prefix paths to generate correlated patterns of higher-order. It has been observed that such an assumption can cause performance problems in CoMine. This paper makes an effort to improve the performance of CoMine by introducing a novel concept known as items' support intervals. The concept says that an item in FP-tree can generate correlated patterns of higher-order by concatenating with only those items in its prefix-paths that have supports within a specific interval. We call the proposed algorithm as CoMine++. Experimental results on various datasets show that CoMine++ can discover high correlated patterns effectively.

**Keywords:** Data mining, Knowledge Discovery in Databases, Correlated patterns and Pattern-growth technique.

## 1 Introduction

Mining frequent patterns [2] from transactional databases has been actively and widely studied in data mining [3, 5]. A major obstacle for the popular adoption of frequent pattern mining in real-world applications is its failure to capture the true correlation relationship among data objects [4]. To confront the obstacle, researchers have made efforts to discover correlated patterns using alternative measures [4, 9, 11, 12]. Although there exists no universally accepted best measure to judge the interestingness of a pattern, *all-confidence* [9] is emerging as a measure that can disclose true correlation relationships among data objects

[13]. An interesting application of correlated patterns is the activity recognition for people with dementia [10]. The model of frequent and correlated patterns is as follows [8].

Let $I = \{i_1, i_2, \cdots, i_n\}$ be a set of items, and $DB$ be a database that consists of a set of transactions. Each transaction $T$ contains a set of items such that $T \subseteq I$. Each transaction is associated with an identifier, called $TID$. Let $X \subseteq I$ be a set of items, referred as an itemset or a *pattern*. A pattern that contains $k$ items is a $k$-pattern. A transaction $T$ is said to contain $X$ if and only if $X \subseteq T$. The support of a pattern $X$ in $DB$, denoted as $S(X)$, is the number of transactions in $DB$ containing $X$. The pattern $X$ is **frequent** if it occurs no less frequent than the user-defined minimum support (*minSup*) threshold, i.e., $S(X) \geq minSup$. The *all-confidence* of a pattern $X$, denoted as *all-conf(X)*, can be expressed as the ratio of its support to the maximum support of an item within it. That is, $all\text{-}conf(X) = \dfrac{S(X)}{max\{S(i_j) | \forall\ i_j \in X\}}$. A pattern $X$ is said to be ***all-confident*** or ***associated*** or ***correlated*** if $S(X) \geq minSup$ and $all\text{-}conf(X) \geq minAllConf$, where *minAllConf* is the user-defined minimum all-confidence threshold.

*Example 1.* Consider the transactional database of 20 transactions shown in Table 1. The set of items $I = \{a,\ b,\ c,\ d,\ e,\ f,\ g,\ h\}$. The set of items $a$ 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 8 transactions (*tids* of $1, 2, 7, 10, 11, 13, 16$ and 19). Therefore, the support of "*ab*," i.e., $S(ab) = 8$. If the user-specified $minSup = 3$, then "*ab*" is a frequent pattern because $S(ab) \geq minSup$. The *all-confidence* of "*ab*", i.e., $all\text{-}conf(ab) = \frac{8}{max(11,9)} = 0.72$. If the user-specified $minAllConf = 0.63$, then "*ab*" is a correlated pattern because $S(ab) \geq minSup$ and $all\text{-}conf(ab) \geq minAllConf$.

**Table 1.** Transactional database.

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

The problem statement of mining correlated patterns is as follows. Given a transactional database ($DB$), and user-defined *minimum support* (*minSup*) and *minimum all-confidence* (*minAllConf*) thresholds, discover the complete set of correlated patterns in $DB$ that satisfy both *minSup* and *minAllConf* thresholds.

CoMine [8] extends FP-growth [6] to discover the complete set of correlated patterns in a database. In particular, CoMine uses pattern-growth technique to discover the patterns. The technique involves the following steps.

*i.* The given database is compressed into a tree known as frequent pattern-tree (FP-tree). The FP-tree retains the pattern association information.

*ii.* Using each item in the FP-tree as an initial suffix item (or pattern)[1], CoMine constructs its Conditional Pattern Base (CPB) consisting of the set of complete prefix paths in the FP-tree co-occurring with the suffix item, then construct its conditional FP-tree and perform mining recursively on such a tree. The pattern-growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from the conditional FP-tree.

The construction of initial CPB for the suffix item in FP-tree is a key step because it reduces the search space effectively. Since the CPB of a suffix pattern represents the set of complete prefix paths in the FP-tree co-occurring with itself, the CoMine implicitly assumes that the suffix item can concatenate with all items in its prefix paths to form correlated patterns of higher-order. This assumption causes performance problems in CoMine. The reason is that *all-confidence* facilitates the suffix item to generate correlated patterns of higher-order by combining with only those items in its prefix-paths that have supports within a specific interval range.

This paper makes an effort to discover correlated patterns effectively. The key contributions of this paper are as follows:

*i.* This paper introduces a novel concept known as items' support intervals. It states that an item can combine with only those items having supports within a specific interval to form correlated (or interesting) patterns of higher-order. A methodology to find items' support intervals for the correlated pattern model has also been discussed in the paper.

*ii.* An improved CoMine, called CoMine++, has also been presented in the paper using items' support intervals. Unlike CoMine, the CPB of the suffix item in CoMine++ represents the set of partial prefix-paths (i.e., involving only those items that have support within the support interval of suffix item) in FP-tree co-occurring with itself.

*iii.* Using the prior knowledge regarding the construction and mining of FP-tree, CoMine++ uses a novel pruning technique to construct the CPB of the suffix item effectively.

*iv.* Experimental results on both synthetic and real-world datasets show that mining high correlated patterns with CoMine++ is time and memory efficient, and highly scalable as well.

The rest of this paper is organized as follows. Section 2 discusses previous works on mining correlated patterns. Section 3 describes the working of CoMine. Section 4 discusses the performance problems of CoMine and introduces CoMine++ to mine correlated patterns. Experimental evaluations of CoMine and CoMine++ are reported in Section 5. Finally, Section 6 concludes with future research directions.

[1] In this paper, the suffix pattern with one item has been referred as suffix item for simplicity purpose.

## 2 Related Work

Since the introduction of frequent patterns in [2], numerous algorithms have been discussed in the literature to mine frequent patterns from transactional databases [5]. FP-growth [6] is a popular algorithm to discover frequent patterns. It uses pattern-growth technique (discussed in Section 1) to discover frequent patterns. The initial CPB of a suffix item in FP-growth represents the set of complete prefix paths in the FP-tree co-occurring with itself. Such a CPB for the suffix item is valid for FP-growth. It is because every suffix item (or pattern) in FP-tree can concatenate with all other items in its prefix-paths to form frequent patterns of higher-order.

Brin et al. [4] have introduced correlated pattern mining using $lift$ and $\chi^2$ as the interestingness measures. Lee et al. [8] have shown that correlated patterns can be effectively discovered with *all-confidence* measure as it satisfies both *null-invariance* and *downward closure properties.* The *null-invariance* property facilitates the measure to disclose genuine correlation relationships without being influenced by the object co-absence in a database. The *downward closure property* facilitates in the reduction of search space as all non-empty subsets of a correlated pattern must also be correlated. An FP-growth-like algorithm known as CoMine was proposed in [8] to discover the patterns using both *all-confidence* and *support* values. The performance issues of CoMine and the techniques to improve the same are discussed in later parts of this paper.

CCMine is a variant of CoMine to discover confidence-closed correlated patterns. Please note that CCMine also suffers from the same performance problems as the CoMine. However, this paper focuses only on CoMine algorithm.

Kim et al. [7] have made an effort to discover top-$k$ correlated patterns using the measures that satisfy null-invariance property. Since some of those measures (e.g. *cosine*) do not satisfy anti-monotonic property, an apriori-like algorithm, NICOMINER, has been proposed by introducing new pruning techniques. Although their work is closely related to our work, their pruning techniques cannot be directly extendable to discover correlated patterns using support and all-confidence measures. The reason is that their work have not discussed any methodology to determine the support intervals for items if correlated patterns are defined using both *support* and *all-confidence* measures. Moreover, NICOMINER suffers from the same performance problems as the Apriori algorithm, which includes the generation of huge number of candidate patterns and multiple scans on the dataset.

CoMine performs better than NICOMINER as it is specifically designed only for *support* and *all-confidence* measures using downward closure property. In the next section, we describe the working of CoMine.

## 3 Working of CoMine

The CoMine uses pattern-growth technique to discover the complete set of correlated patterns in a database. The technique involves: ($i$) compressing the database into FP-tree and (*ii*) recursive mining of FP-tree to discover the complete set of correlated patterns.

The FP-tree is a compact data structure for storing all necessary information about frequent patterns in a database. Every branch of the FP-tree represents a pattern, and the nodes along the branch are ordered decreasingly by the frequency of the corresponding item, with leaves representing the least frequent items. Compression is achieved by building the tree in such a way that overlapping patterns are represented by sharing pre-fixes of the corresponding branches. It has a header table containing two fields: item name ($I$), and their support count ($S$). A row in the header table also contains the head of a list that links all the corresponding nodes of the FP-tree.

The FP-tree is constructed with only two scans on the database. In the first scan, all frequent items are found and sorted in support descending order. Using the sorted order, the second scan constructs the FP-tree which stores all frequency information of the original dataset. Mining the database then becomes mining the FP-tree. For the user-defined $minSup = 3$ and $minAllConf = 0.63$, the set of sorted frequent items found after first scan on the database of Table 1 are $\{\{a : 11\}, \{b : 9\}, \{c : 9\}, \{d : 6\}, \{e : 5\}, \{f : 5\}\}$. Figure 1 shows the FP-tree constructed after performing the second scan on the database.

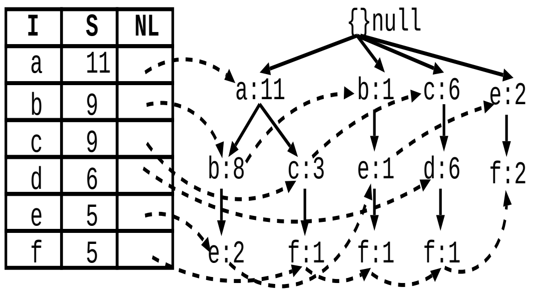

**Fig. 1.** FP-tree. The terms 'I', 'S' and 'NL' respectively denote item, support and node-link.

**Table 2.** Mining correlated patterns with CoMine.

| Suffix item | Conditional Pattern Base | Conditional FP-tree | Correlated Patterns |
|---|---|---|---|
| $f$ | $\{\{ac : 1\}$, $\{cd : 1\}$, $\{e : 2\}$, $\{be : 1\}\}$ | $\langle e : 3\rangle$ | {ef : 3} |
| $e$ | $\{ab : 2\}$ | - | - |
| $d$ | $\{c : 6\}$ | $\langle c : 6\rangle$ | {cd : 6} |
| $c$ | $\{a : 3\}$ | - | - |
| $b$ | $\{ea : 8\}$ | $\langle a : 8\rangle$ | {ab : 8} |

Mining correlated patterns using FP-tree of Figure 1 is shown in Table 2 and detailed as follows. Consider the item $f$, which is the last item in FP-tree, as a suffix item. The item $f$ occurs in four branches of the FP-tree of Figure

1. The paths formed by these branches are $\langle acf : 1\rangle$, $\langle cdf : 1\rangle$ $\langle ef : 2\rangle$ and $\langle bef : 1\rangle$. Therefore, considering $f$ as a suffix, its corresponding four prefix paths are $\langle ac : 1\rangle$, $\langle cd : 1\rangle$, $\langle e : 2\rangle$ and $\langle be : 1\rangle$, which form its conditional pattern base. Its conditional FP-tree contains only a single path, $\langle e : 3\rangle$. The items $a$, $b$, $c$ and $d$ are not included in Conditional FP-tree because their support of 1 is less than $minSup$ (also, the all-confidence values of $af$, $bf$, $cf$ and $df$ are less than $minAllConf$). The concatenation of suffix pattern with conditional FP-tree generates the correlated pattern $\{ef : 3\}$. Similar process is repeated for other items in the FP-tree to discover the complete set of correlated patterns.

In the next section, we discuss the performance issues of CoMine and propose techniques to improve its performance.

## 4 Proposed Algorithm

In this section, we first introduce the concept of items' support intervals. This term simplifies the understanding of performance issue in CoMine, which is discussed subsequently. Next, we propose the basic idea of incorporating the proposed concept in CoMine to improve its performance. We call the improved CoMine as CoMine++.

### 4.1 Items' Support Intervals

The concept of *items' support intervals* say that every item can generate interesting patterns by combining with only those items having supports within a specific interval. Finding the support interval for each item in the database is non-trivial because it depends upon the following two factors: (*i*) the support of an item and (*ii*) the user-defined threshold values' for the measures. In this paper, we make an effort to identify the items' support interval for correlated pattern model defined using *support* and *all-confidence* measures.

For the user-defined $minAllConf$ and $minSup$ thresholds, the support interval of an item $i_j \in I$ is $\left[ max\begin{pmatrix} S(i_j) \times minAllConf, \\ minSup \end{pmatrix}, max\begin{pmatrix} \frac{S(i_j)}{minAllConf}, \\ minSup \end{pmatrix}\right]$. The correctness is shown in Theorem 1, and is based on Properties 1 and 2 and Lemmas 1 and 2.

*Property 1.* If $X$ and $Y$ are two patterns such that $X \subset Y$, then $S(X) \geq S(Y)$.

*Property 2.* The maximum support a pattern $X$ can have is $min(S(i_j)|\forall i_j \in X)$. Therefore, the maximum all-confidence a pattern $X$ can have is $\frac{min(S(i_j)|\forall i_j \in X)}{max(S(i_j)|\forall i_j \in X)}$.

**Lemma 1.** *Let $minAllConf$ be the user-defined minimum all-confidence threshold value. The lower limit of a support interval for an item $i_q \in I$ having support $S(i_q)$ is $S(i_q) \times minAllConf$.*

*Proof.* Let '$i_p$' and '$i_q$' be the two items (or 1-patterns) having supports such that $S(i_p) < S(i_q)$. The maximum all-confidence the pattern '$i_p i_q$' (=$\{i_p,\ i_q\}$) can have is $\frac{S(i_p)}{S(i_q)}$ (Property 2). If $S(i_q) \times minAllConf > S(i_p)$, then $minAllConf > \frac{S(i_p)}{S(i_q)}$ $(= \textit{all-conf}(i_p i_q))$. If $S(i_q) \times minAllConf = S(i_p)$, then $minAllConf = \frac{S(i_p)}{S(i_q)}$ $(= \textit{all-conf}(i_p i_q))$. Therefore, the lower limit of a support interval for an item $i_q \in I$ with support $S(i_q)$ is $S(i_q) \times minAllConf$.

**Lemma 2.** *Let minAllConf be the user-defined minimum all-confidence threshold value. The upper limit of a support interval for an item* $i_q \in I$ *having support* $S(i_q)$ *is* $\dfrac{S(i_q)}{minAllConf}$.

*Proof.* Let '$i_q$' and '$i_r$' be the two items (or 1-patterns) having supports such that $S(i_q) < S(i_r)$. Using Property 2, the maximum all-confidence for the pattern '$i_q i_r$' (=$\{i_q,\ i_r\}$) can be $\frac{S(i_q)}{S(i_r)}$. If $S(i_r) > \dfrac{S(i_q)}{minAllConf}$, then $minAllConf > \dfrac{S(i_q)}{S(i_r)}$ $(= \textit{all-conf}(i_q i_r))$. Therefore, the upper limit of a support interval for an item $i_q \in I$ is $\dfrac{S(i_q)}{minAllConf}$.

**Theorem 1.** *For the user-defined minSup and minAllConf thresholds, the support interval for an* $i_q \in I$ *with support* $S(i_q)$ *is*

$$\left[ max\begin{pmatrix} S(i_q) \times minAllConf, \\ minSup \end{pmatrix},\ max\begin{pmatrix} \dfrac{S(i_q)}{minAllConf}, \\ minSup \end{pmatrix} \right].$$

*Proof.* The support interval for an item $i_q \in I$ for *support* measure is $[minSup,\ \infty)$. In other words, items having support no less than *minSup* can combine with one another in all possible ways to generate frequent patterns of higher-order. Using Lemmas 1 and 2, the support interval of an item $i_q \in I$ for *all-confidence* measure is $\left[S(i_q) \times minAllConf,\ \dfrac{S(i_q)}{minAllConf}\right]$. Therefore, the support interval for an item $i_q \in I$ for both *support* and *all-confidence* measures is

$$\left[ max\begin{pmatrix} S(i_q) \times minAllConf, \\ minSup \end{pmatrix},\ max\begin{pmatrix} \dfrac{S(i_q)}{minAllConf}, \\ minSup \end{pmatrix} \right].$$

*Example 2.* The support of $f$ in Table 1 is 5. If $minAllConf = 0.63$ and $minSup = 3$, then $f$ can generate correlated patterns by combining with only those items having frequencies in the range of [3, 8] $(= [max(5 \times 0.63, 3), max(\frac{5}{0.63}, 3)])$.

## 4.2 Performance Problems in CoMine

Since the initial CPB of a suffix item constitutes of the set of complete prefix paths in the FP-tree co-occurring with itself, CoMine implicitly assumes that the suffix item can generate correlated patterns of higher-order by concatenating with all items in its prefix path. However, this is not the seldom case because *all-confidence* facilitates the suffix item to concatenate with only those items in its prefix paths that have support within a specific interval range to generate the patterns. As a result, CoMine suffers from the performance problems pertaining to both memory and runtime requirements.

*Example 3.* Since the CPB of $f$ contains $a$, $b$ and $c$, CoMine implicitly assumes that $f$ can concatenate with $a$, $b$ and $c$ items to generate correlated patterns. However, any correlated pattern containing $f$ can never have the items, $a$, $b$ and $c$. It is because their supports do not lie within the support interval of $f$.

## 4.3 Basic Idea: Pruning Technique

Using the concept of items' support intervals, the complete set of correlated patterns can be discovered from FP-tree by building the CPB of the initial suffix item with the set of partial prefix paths involving only those items that have supports within its support interval.

To construct such CPB for a suffix item, one can assume that the support of every item present in the prefix path of the suffix item needs to tested, which is same as constructing CPB with complete set of prefix paths (as in CoMine). This paper argues that such a process is not necessary. It is because of the following pruning technique. *If an item at level k in the prefix path of the suffix item fails to have its support within the support interval of corresponding suffix item, then all items lying in between the root node and the failed item (i.e., items lying in levels 1 to* $k-1$*) will also fail to have their supports within the support interval of corresponding suffix item.* The correctness is shown in Lemma 3 and illustrated in Example 4.

**Lemma 3.** *Let* $\langle i_1, i_2, \cdots, i_k, i_{k+1}\rangle$, $1 \leq k < n$, *be a branch in FP-tree such that* $S(i_1) \geq S(i_2) \geq \cdots \geq S(i_k) \geq S(i_{k+1})$. *If* $all\text{-}conf(i_k i_{k+1}) < minAllConf$, *then* $all\text{-}conf(i_j i_{k+1}) < minAllConf$, *where* $1 \leq j < k$.

*Proof.* The prefix path for the suffix item $i_{k+1}$ is $\langle i_1, i_2, \cdots, i_k\rangle$. If $all\text{-}conf(i_k, i_{k+1}) < minAllConf$, then $S(i_k) > \dfrac{S(i_{k+1})}{minAllConf}$ (using Lemma 2). Since FP-tree is constructed in support descending order of items, it turns out that $S(i_j) \geq S(i_k) > \dfrac{S(i_{k+1})}{minAllConf}$, where $1 \leq j < k$. In other words, $all\text{-}conf(i_j i_{k+1}) < minAllConf$, $1 \leq j < k$.

*Example 4.* A prefix path for the suffix item $f$ is $\langle a, c : 1\rangle$. The support of $c$ does not lie in the support interval of $f$, i.e., $S(c) > \dfrac{S(f)}{minAllConf}$. As a result, it can be stated without testing that the support of $a$ will not lie in the support interval of $f$, i.e., $S(a) > \dfrac{S(f)}{minAllConf}$.

### 4.4 CoMine++ Algorithm

The proposed CoMine++ involves two steps: (*i*) Construction of FP-tree and (*ii*) Mining FP-tree to discover the complete set of correlated patterns. The first step is same as in CoMine. However, the second step is different, and is as follows. Considering each frequent item in the FP-tree as a suffix item, CoMine++ constructs its CPB with the set of partial prefix paths containing only those items having supports within its support interval. Next, compress the CPB by performing tree-merging operations, which involves merging the common nodes in the CPB by summing up their respective supports. Next, conditional FP-tree is built using $minSup$ and $minAllConf$ constrains. The conditional FP-tree contains only those items that have generated correlated patterns by concatenating with the suffix item. Thus, subsequent recursive mining on conditional FP-tree involves constructing CPBs with the set of complete prefix paths co-occurring with the suffix pattern. The pattern-growth is achieved by the concatenation of the suffix pattern with the correlated patterns generated from a conditional FP-tree.

The working of CoMine++ is shown in Algorithm 1, Procedure 2 and Procedure 3. The input parameters to CoMine++ are transactional database and the user-defined $minSup$ and $minAllConf$ thresholds. The Algorithm 1 constructs FP-tree using $minSup$ threshold and calls Procedure 2 for mining correlated patterns from FP-tree. Procedure 2 builds CPB for the suffix item with the set of partial prefix paths involving only those items having supports within its support interval, performs *tree-merging* operation to compress the CPB, constructs conditional FP-tree for the suffix item and calls Procedure 3 for recursive mining of conditional FP-tree of the suffix item. The working of Procedure 3 resembles Procedure 2. However, the variation is that Procedure 3 builds the CPB of the suffix pattern with the complete set of prefix paths.

We now explain the working of CoMine++ using the database shown in Table 1. Let $minSup = 3$ and $minAllConf = 0.63$. First, FP-tree shown in Figure 1 is created using $minSup = 3$. Next, mining correlated patterns using FP-tree is shown in Table 3 and detailed as follows. Consider the item $f$ as a **suffix item**. The support interval of $f$ is [3, 7]. The item $f$ occurs in four branches of the FP-tree of Figure 1. The paths formed by these branches are $\langle acf : 1\rangle$, $\langle cdf : 1\rangle$ $\langle ef : 2\rangle$ and $\langle bef : 1\rangle$ (see Figure 2(a)). Therefore, considering $f$ as a suffix, its corresponding four prefix paths are $\langle ac : 1\rangle$, $\langle cd : 1\rangle$, $\langle e : 2\rangle$ and $\langle be : 1\rangle$ (see Figure 2(b)). In the prefix path $\langle ac : 1\rangle$, the item $c$ fails the test (i.e., its support does not lie in the support interval of the suffix item $f$). Using the pruning technique discussed in the basic idea, we neglect testing of $a$ in the prefix path (line 6 in Procedure 2) and directly select another prefix path for testing. In the prefix path $\langle cd : 1\rangle$, the item $d$ satisfies the test. As a result, the item $c$ is tested and is found that it fails the test. Similar process is repeated for the other prefix paths of $f$. Finally, we neglect the items $a$, $b$ and $c$ and construct CPB of $f$ with only the items $d$ and $e$, resulting in three branches $\langle e : 1\rangle$, $\langle d : 1\rangle$ and $\langle e : 2\rangle$ (see Figure 2(c)). Since two branches in the CPB of $f$ contain a common item $e$, tree-merging operation (line 8 in Procedure 2) is performed on the CPB of $f$ to

```
Algorithm 1 CoMine++
INPUT: Transactional database $DB$, minimum support ($minSup$) and minimum all-confidence ($minAllConf$)
OUTPUT: Complete set of correlated patterns.
1: The FP-tree is constructed in the following steps:
     i. Scan the transactional database $D$ once. Collect $F$, the set of frequent items, and their support counts. Sort $F$ in support descending order as $L$, the list of frequent items.
     ii. Create the root of an FP-tree, and label it as "null." For each transaction $t$ in $D$ do the following. Select and sort the frequent items in $t$ according to the order of $L$. Let the sorted frequent item list in $t$ be $[p|P]$, where $p$ is the first element and $P$ is the remaining list. Call insert_tree($[p|P], T$), which is performed as follows. If $T$ has a child $N$ such that $N.item\text{-}name = p.item\text{-}name$, then increment $N$'s count by 1; else create a new node $N$, and let its count be 1, its parent link be linked to $T$, and its node-link to the nodes with the same item-name via the node-link structure. If $P$ is non-empty, call insert_tree($P, N$) recursively.
2: The FP-tree is mined by calling Co-mine_1($Tree, null$).
```

create two branches $\langle d : 1\rangle$ and $\langle e : 3\rangle$ (see Figure 2(d)). The conditional FP-tree of $f$ contains only a single path, $\langle e : 3\rangle$, $d$ is not included because its support of 1 is less than *minSup*. The concatenation of suffix pattern with conditional FP-tree generates the correlated pattern $\{ef : 3\}$. Similar process is repeated for other items in the FP-tree to discover the complete set of correlated patterns.

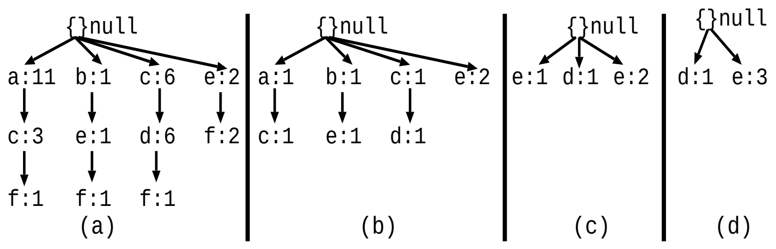

**Fig. 2.** Generating conditional pattern base of $f$. (a) Branches of FP-tree containing $f$, (b) prefix paths of $f$, (c) partial prefix paths of $f$ and (d) Final conditional pattern base of $f$ after merging its partial prefix paths.

## 5 Experimental Results

In this section, we evaluate the perform of CoMine and CoMine++ algorithms. We show that CoMine++ is a better algorithm to mine highly correlated patterns in different types of datasets. The algorithms are written in GNU C++ and run with the Ubuntu 10.04 operating system on a 2.66 GHz machine with 1GB

**Procedure 2** Co-mine_1($Tree, \alpha$); Constructing the conditional pattern base for frequent item or length-1 suffix pattern.

```
1: for each $a_i$ in the header of $Tree$ do
2:    Generate pattern $\beta = \alpha \cup a_i$ with $all\text{-}conf = \frac{S(\beta)}{max\_item\_sup(\beta)}$. {$S(\beta) = S(a_i)$ in $\alpha$-projected database}
3:    Construct $\beta$-projected database as follows.
4:    for each prefix path $PP_k$ of $a_i$ do
5:       Starting from the last item in $PP_k$ and moving up the prefix path, test whether their support is no less than $max(\frac{S(a_i)}{minAllConf}, minSup)$.
6:       Skip testing other items in $PP_k$ if an item at level $l$ in $PP_k$ fails the test. It is because other items lying in between the root node and the failed item in $PP_k$ will fail the test.
7:       Create a temporary branch involving only those items in $PP_k$ that satisfy the test. Preserve the node counts of the items and their order in the temporary branch.
8:       Add the temporary branch in $\beta$-projected database. This step is similar to insert_tree function in Procedure 1. Reduce the size of $\beta$-projected database by merging the common nodes and summing up their node counts.
9:    end for{CoMine do not perform the above steps (i.e., lines from 4 to 9). It simply constructs $\beta$-projected database with every item in a prefix path of $\beta$.}
10:   Let $I_\beta$ be the set of items in $\beta$-projected database.
11:   For each item in $I_\beta$, compute its count in $\beta$-projected database;
12:   for each $b_j \in I_\beta$ do
13:      if $S(\beta \cup b_j) < minSup$ then
14:         delete $b_j$ from $I_\beta$; {pruning based on minimum support}
15:      end if
16:      if $all\text{-}conf(\beta \cup b_j) < minAllConf$ then
17:         delete $b_j$ from $I_\beta$; {pruning based on minimum all-confidence}
18:      end if
19:   end for
20:   construct $\beta$-conditional FP-tree with items in $I_\beta$ $Tree_\beta$.
21:   if $Tree_\beta \neq \emptyset$ then
22:      Co-mine_k($Tree_\beta, \beta$);
23:   end if
24: end for
```

**Table 3.** Mining correlated patterns with CoMine++.

| Item | Conditional Pattern Base | Conditional FP-tree | Correlated Patterns |
|---|---|---|---|
| $f$ | $\{\{d : 1\}, \{e : 3\}\}$ | $\langle e : 3\rangle$ | {ef : 3} |
| $e$ | - | - | - |
| $d$ | $\{c : 6\}$ | $\langle c : 6\rangle$ | {cd : 6} |
| $c$ | $\{a : 3\}$ | - | - |
| $b$ | $\{a : 8\}$ | $\langle a : 8\rangle$ | {ab : 8} |

**Procedure 3** Co-mine_k($Tree, \alpha$); Constructing the conditional pattern base for length-$k$ suffix pattern, where $k \geq 2$.

```
 1: for each $a_i$ in the header of $Tree$ do
 2:   Generate pattern $\beta = \alpha \cup a_i$ with all-conf $= \frac{S(\beta)}{max\_item\_sup(\beta)}$. {$S(\beta) = S(a_i)$ in $\alpha$-projected database}
 3:   Get a set $I_\beta$ of items to be included in $\beta$-projected database. {No need to check whether $S(i_j) \leq \frac{min\_item\_sup(\beta)}{minAllConf}$ }
 4:   for each item in $I_\beta$, compute its count in $\beta$-projected database;
 5:   for each $b_j \in I_\beta$ do
 6:     if $S(\beta \cup b_j) < minSup$ then
 7:       delete $b_j$ from $I_\beta$; {pruning based on minimum support}
 8:     end if
 9:     if all-conf$(\beta \cup b_j) < minAllConf$ then
10:       delete $b_j$ from $I_\beta$; {pruning based on minimum all-confidence}
11:     end if
12:   end for
13:   construct $\beta$-conditional FP-tree with items in $I_\beta$ $Tree_\beta$.
14:   if $Tree_\beta \neq \emptyset$ then
15:     Co-mine_k($Tree_\beta, \beta$);
16:   end if
17: end for
```

memory. The runtime is expressed in seconds and specifies the total execution time, i.e., CPU and I/Os. We pursued experiments on synthetic (T10I4D100K) and real-world (BMS-WebView-1,BMS-WebView-2, Mushroom and Kosarak) datasets. The datasets are available at Frequent Itemset Mining repository [1]. The details of these datasets are shown in Table 4.

Please note that we are not comparing the proposed CoMine++ with the NICOMINER [7]. It is because of two reasons: (*i*) CoMine performs better than the NICOMINER as the latter suffers from the same performance problems as the Apriori and (*ii*) NICOMINER is meant to discover top-k correlated patterns and not the complete set of correlated patterns in a database.

**Table 4.** Dataset Characteristics. The terms “Max.”, “Avg.” and “Tran.” are respectively used as the acronyms for “maximum”, “average” and “transaction.“

| Dataset | Transactions | Distinct Items | Max. Trans. Size | Avg. Trans. Size | Type |
|---|---|---|---|---|---|
| *T10I4D100k* | 100000 | 870 | 29 | 10.102 | sparse |
| *BMS-WebView-1* | 59602 | 4971 | 267 | 2.5 | sparse |
| *BMS-WebView-2* | 77512 | 33401 | 161 | 2 | sparse |
| *Mushroom* | 8124 | 119 | 23 | 23.0 | dense |
| *Kosarak* | 990002 | 41270 | 2498 | 8.1 | sparse |

## 5.1 Memory Tests on CoMine and CoMine++ Algorithms

Figure 3 (a), (b), (c) and (d) respectively show the number of nodes generated by CoMine and CoMine++ algorithms at different *minAllConf* values in T10I4D100k, BMS-WebView-1, BMS-WebView-2 and Mushroom datasets, respectively. The *minSup* used in these datasets are 0.1%, 0.1%, 0.1% and 25%, respectively. CoMine++ generates relatively fewer numbers of nodes than CoMine with the increase in *minAllConf* value. It is because of the decrease in the width of items' support intervals with the increase in *minAllConf* value.

Since the memory requirement of an algorithm depends on the number of nodes being generated by an algorithm, it turns out that CoMine++ is more memory efficient than CoMine.

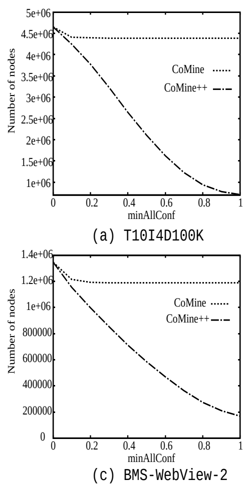

**Fig. 3.** Number of nodes generated in CoMine and CoMine++ algorithms.

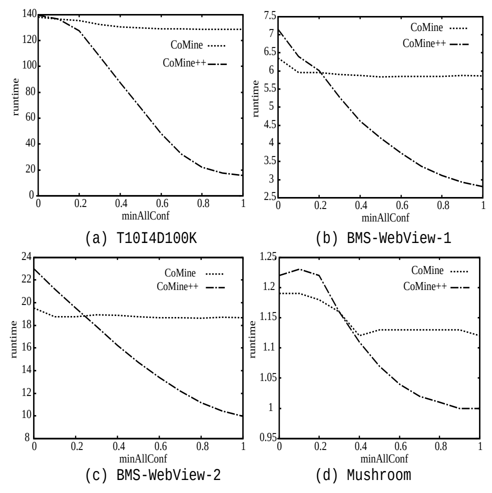

**Fig. 4.** Runtime (in seconds) comparison of CoMine and CoMine++ algorithms.

## 5.2 Runtime Tests on CoMine and CoMine++ Algorithms

Figure 4 (a), (b), (c) and (d) respectively show the runtime taken by CoMine and CoMine++ algorithms at different *minAllConf* values in T10I4D100k, BMS-WebView-1, BMS-WebView-2 and Mushroom datasets, respectively. The *minSup* used in these datasets are 0.1%, 0.1%, 0.1% and 25%, respectively. The following three observations can be drawn from these graphs.

- Increase in *minAllConf* has decreased the runtime in both CoMine and CoMine++ algorithms. It is because of decrease in the number of correlated patterns with the increase in *minAllConf* value.
- CoMine++ has outperformed CoMine at higher *minAllConf* values. It is because of the decrease in items' support interval which resulted in the construction of the initial CPB of a suffix item timely.

– CoMine has performed better than CoMine++ at low *minAllConf* values. It is because of increase in items' support interval which resulted in CoMine++ to construct the CPB of the suffix item with almost all items in its prefix paths.

### 5.3 Scalability Test on CoMine and CoMine++ Algorithms

In this experiment, we evaluate the scalability performance of CoMine and CoMine++ algorithms on memory and runtime requirements 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 ten portions of 0.1 million transactions in each part. Then we investigated the performance of CoMine and CoMine++ algorithms after accumulating each portion with previous parts while performing correlated pattern mining each time. We fixed $minSup = 0.1\%$ and $minAllConf = 0.5$ for each experiment. The experimental results are shown in Figure 5. The memory (represented in number of nodes) and time in $y$-axes of the left and right graphs in Figure 5 respectively specify the required memory and total runtime with the increase of database size. It is clear from the graphs that as the database size increases, overall tree construction and mining time, and required memory increase. However, CoMine++ requires relatively less memory (nearly half of the memory requirements of CoMine) and runtime with respect to the database size. Therefore, it can be observed from the scalability test that CoMine++ can efficiently mine correlated patterns over large datasets and distinct items with considerable amount of runtime and memory.

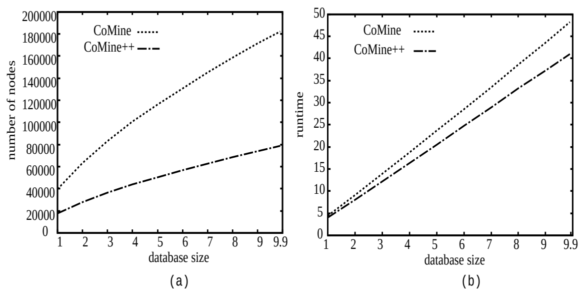

**Fig. 5.** Scalability test. (a) Memory requirement and (b) Runtime.

In many real-world applications, users are generally interested in highly correlated patterns. Thus, the proposed CoMine++ algorithm is a better choice over the existing CoMine algorithm.

## 6 Conclusion

This paper argues that a pattern-growth algorithm that simply constructs the CPB of the suffix item with every item in its prefix path can suffer from performance problems. It is because some measures facilitate an item to combine with only those items having supports within a specific interval to generate interesting patterns of higher order. This paper introduced the concept of items' support intervals and proposed a methodology to determine it for the correlated pattern model defined using *support* and *all-confidence* measures. A pattern-growth algorithm, called CoMine++, has also been proposed to discover correlated patterns effectively. Unlike the traditional pattern-growth algorithms (such as CoMine), CoMine++ discovers the complete set of correlated patterns by constructing the initial CPB of the suffix item with only those items in its prefix paths that have support within its support interval. A novel pruning technique has also been discussed to construct CPB of the suffix item effectively. Experimental results have shown that proposed CoMine++ algorithm can efficiently mine highly correlated patterns over the existing CoMine algorithm.

As a part of future work, we would like to extend the notion of items' support intervals to improve the performance of CCMine algorithm to discover closed coverage patterns effectively. In addition, we would like to investigate the methods to determine the items' support intervals for other measures.

## References

1. Frequent itemset mining repository. `http://fimi.cs.helsinki.fi/data/`.
2. R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93*, pages 207–216. ACM, 1993.
3. R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. VLDB '94, pages 487–499, 1994.
4. S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: generalizing association rules to correlations. *SIGMOD Rec.*, 26(2):265–276, 1997.
5. J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: Current status and future directions. *Data Mining and Knowledge Discovery*, 14(1), 2007.
6. 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.
7. S. Kim, M. Barsky, and J. Han. Efficient mining of top correlated patterns based on null-invariant measures. In *Proceedings of the 2011 European conference on Machine learning and knowledge discovery in databases - Volume Part II*, ECML PKDD'11, pages 177–192, Berlin, Heidelberg, 2011. Springer-Verlag.
8. Y. K. Lee, W. Y. Kim, D. Cai, and J. Han. Comine: efficient mining of correlated patterns. pages 581 – 584, nov. 2003.
9. E. R. Omiecinski. Alternative interest measures for mining associations in databases. *IEEE Trans. on Knowl. and Data Eng.*, 15(1):57–69, 2003.
10. K. Sim, C. Phua, G. Yap, J. Biswas, and M. Mokhtari. Activity recognition using correlated pattern mining for people with dementia. *Conf Proc IEEE Eng Med Biol Soc*, 2011, 2011.
11. P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *KDD '02*, pages 32–41, New York, NY, USA, 2002. ACM.
12. T. Wu, Y. Chen, and J. Han. Re-examination of interestingness measures in pattern mining: a unified framework. *Data Min. Knowl. Discov.*, 21(3):371–397, 2010.
13. W. young Kim, Y.-K. Lee, and J. Han. Ccmine: Efficient mining of confidence-closed correlated patterns. In *In PAKDD*, pages 569–579, 2004.