Towards Efficient Discovery of Frequent Patterns with Relative Support

R. Uday Kiran and Masaru Kitsuregawa
Institute of Industrial Science,
University of Tokyo, Japan.
Email: uday_rage@tkl.iis.u-tokyo.ac.jp and kitsure@tkl.iis.u-tokyo.ac.jp.

ABSTRACT

Frequent patterns are an important class of regularities that exist in a database. Although there exists no universally acceptable best measure to assess the interestingness of a pattern, *relative support* is emerging as a popular measure to discover frequent patterns involving both frequent and rare items. An Apriori-like algorithm known as Relative Support Apriori (RSA) has been discussed in the literature to discover the patterns. It has been observed that mining frequent patterns with RSA is a computationally expensive process because the discovered patterns do not satisfy the anti-monotonic property. Moreover, RSA also suffers from the performance problems involving generation of the huge number of candidate patterns and multiple scans on the database. This paper makes an effort to discover frequent patterns effectively with the *relative support* measure. To reduce the computational cost, we theoretically show that the patterns discovered with the *relative support* measure satisfy the convertible anti-monotonic property. Using this property, a pattern-growth algorithm known as Relative Support Frequent Pattern-growth (RSFP-growth) has been proposed to discover the patterns. Experimental results on both synthetic and real-world datasets show that the proposed RSFP-growth algorithm is significantly better than the RSA algorithm.

1. INTRODUCTION

Frequent pattern mining is an important knowledge discovery technique in data mining. In the basic model of frequent patterns [3], a pattern (or an itemset) is considered frequent if it satisfies the user-defined minimum support (*minsup*) constraint. The *minsup* constraint controls the minimum number of transactions a pattern must cover in a database. Since only a single *minsup* constraint is used for the entire dataset, the model implicitly assumes that all items in a database have uniform frequencies. However, this is often not the case in many real-world databases. In many real-world applications, some items appear very frequently in the data, while others rarely appear. It has to be noted that considering an item in a database as either frequent or rare is a subjective issue which depends on the user and/or application requirements. If the items' frequencies in a database vary widely, we encounter the following issues:

i. If *minsup* is set too high, we will miss those patterns that involve rare items.

ii. In order to find the 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 can be meaningless depending upon the user and/or application requirements.

This dilemma is called the *rare item problem* [19]. One cannot ignore the knowledge pertaining to rare items. It is because such knowledge has been found useful in many real-world applications, such as detecting oil spills in satellite images [11] and improving the performance of recommender systems [5].

To confront the rare item problem in applications, numerous alternative measures have been discussed in the literature [4, 13, 18, 20, 21]. Unlike the *support* measure, these measures assess the interestingness of a pattern with respect to the frequencies of items within it. Each measure has a selection bias that justifies the significance of a knowledge pattern. As a result, there exists no universally acceptable best measure to judge the interestingness of a pattern for any given dataset or application. Researchers are making efforts to suggest a right measure depending upon the user and/or application requirements [17, 18, 20].

Yun et al. [21] have introduced the *relative support* measure and showed that it can discover the frequent patterns involving significant rare items effectively. A significant rare item represents an item that appears less frequently in the database (or having *support* less than the user-defined *minsup* threshold) but appears associated with the frequently occurring items in high proportion to its frequency. An Apriori-like algorithm known as Relative Support Apriori (RSA) algorithm has also been introduced to discover the patterns. We have observed that RSA is a computationally expensive algorithm because the patterns discovered with the *relative support* measure do not satisfy the anti-monotonic property. Moreover, RSA also suffers from the performance problems involving generation of the huge number of candidate patterns and multiple scans on the database.

This paper makes an effort to discover frequent patterns efficiently using the *relative support* and *support* measures. The contributions are as follows.

- The paper theoretically investigates the *relative support* measure and shows that the discovered patterns satisfy a property known as the *convertible anti-monotonic property* [15].

The 18th International Conference on Management of Data (COMAD), 14th-16th Dec, 2012 at Pune, India.

- Using the convertible anti-monotonic property and prior knowledge regarding the FP-growth construction and mining technique, we redefine the frequent pattern with a novel concept known as the *conditional minimum support*.
- Using the conditional minimum support, an FP-growth-like algorithm known as Relative Support Frequent Pattern-growth (RSFP-growth) has been proposed in this paper. Pei et al. [15] have theoretically shown that the search space for a pattern-growth algorithm discovering interesting patterns satisfying the convertible anti-monotonic property is same as the search space of a pattern-growth algorithm discovering interesting patterns satisfying the anti-monotonic property. Therefore, it can be said that the RSFP-growth algorithm reduces the computational cost of mining the patterns effectively.
- Experimental results on various datasets show that the RSFP-growth algorithm is runtime efficient and scalable than the RSA algorithm.

The rest of this paper is organized as follows. Section 2 discusses previous works on mining frequent patterns. Section 3 introduces the model of frequent patterns using the *relative support* and *support* measures. Section 4 discusses the performance issues of RSA algorithm, describes the basic idea and introduces the proposed RSFP-growth algorithm. Experimental evaluations of RSA and RSFP-growth algorithms are presented in Section 5. Finally, Section 6 concludes with future research directions.

2. RELATED WORK

Since the usage of single minimum support (*minsup*) framework to discover frequent patterns involving both frequent and rare items suffers from the dilemma known as the rare item problem, researchers have made efforts to discover frequent patterns with multiple *minsups* framework [7, 8, 9, 10, 12, 16], or other interestingness measures [4, 13, 18, 20, 21], or constraints [14].

In the multiple *minsups* framework, the user specifies a *minsup*-like constraint, called *minimum item support*, for every item in the database. Next, the *minsup* for a pattern is represented with the minimum item supports of the items within it. An open research problem of this framework is the methodology to determine the *minimum item supports* of all the items in a database.

Researchers have also introduced numerous alternative measures to discover frequent patterns. Examples includes *all-confidence*, *any-confidence*, *bond* [13], $lift$ and χ^2 [4]. Each measure has its own selection bias that justifies the significance of knowledge pattern. Thus, there exists no universally acceptable best measure to judge the interestingness of a pattern for any given application. In other words, selecting a right measure to discover frequent patterns involving both frequent and rare items is an important research issue. Researchers are making efforts to suggest a right algorithm depending upon the user and/or application requirements [17, 18, 20].

Yun et al. [21] have introduced the *relative support* measure to discover the frequent patterns involving significant rare items. An Apriori-like algorithm known as RSA has also been proposed to discover the complete set of patterns with the *relative support* and *support* measures [21]. The pattern model used in RSA requires three user-specified thresholds: first *support* (s_1), second *support* (s_2) such that $s_1 > s_2$ and minimum *relative support* (*minRsup*). The s_1 and s_2 thresholds are used to classify the items into either frequent or rare items. That is, items having *support* no less than s_1 are classified as **frequent items** and the items having *support* in between s_1 and s_2 are classified as **significant rare items**. If a pattern contains only frequent items, its *support* has to satisfy s_1 to be a frequent pattern. If an pattern contains rare items, its *support* must satisfy s_2 and its *relative support* must satisfy *minRsup* to be a frequent pattern. In many real-world applications, it is often difficult for the user to classify the items into either frequent or rare items. Therefore, a simplified approach is discover frequent itemsets using only s_2 and *minRsup* thresholds. In this paper, we use this approach to discover the frequent patterns. The performance issues of RSA algorithm and our efforts to address them are discussed in later parts of this paper.

3. THE FREQUENT PATTERN MODEL USING RELATIVE SUPPORT

The frequent pattern model using the *relative support* measure is as follows [21]:

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 frequency (or *support* count) of a pattern X in *DB*, denoted as $f(X)$, is the number of transactions in *DB* containing X. The *support* of X, denoted as $S(X)$, is the ratio of its frequency to the *DB* size, i.e., $S(X) = \frac{f(X)}{|DB|}$. The *relative support* of a pattern X, denoted as $Rsup(X)$, is the ratio of its *support* to the minimum *support* of an item (or 1-pattern) within it. That is, $Rsup(X) = \frac{S(X)}{min(S(i_j)|\forall i_j \in X)}$. The pattern X is **frequent** if its *support* and *relative support* are no less than the user-defined minimum *support* (*minsup*) and minimum *relative support* (*minRsup*) thresholds. That is, X is said to be frequent if

$$\begin{array}{rcl} S(X) & \geq & minsup \\ & and & \\ Rsup(X) & \geq & minRsup. \end{array} \quad (1)$$

We call the frequent pattern containing only one item as **frequent item**. We now illustrate the frequent pattern model using the transactional database shown in Table 1.

Table 1: Transactional database.

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

EXAMPLE 1. *The transactional database shown in Table 1 has 20 transactions. The set of items* $I = \{a,b,c,d,e,f,g\}$. *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 count of "ab," i.e.,* $f(ab) = 8$. *The* support *of ab, i.e.,* $S(ab) = 0.4 \left(= \frac{8}{20}\right)$.

The relative support *of ab, i.e.,* $Rsup(ab) = 0.88 \left(= \frac{0.4}{min(0.6, 0.45)}\right)$. *If the user-specified* $minsup = 0.3$ *and* $minRsup = 0.65$*, then ab is a frequent pattern because* $S(ab) \geq minsup$ *and* $Rsup(ab) \geq minRsup$.

It can be observed that the *relative support* measure assess the interestingness of a pattern with respect to the minimal *support* of an item within it. Thus, it can effectively discover all those frequent patterns that contain rare items occurring together with other items in high proportions of their frequencies.

The problem definition is as follows. Given the transactional database (*DB*), and the user-defined minimum *support* (*minsup*) and minimum *relative support* (*minRsup*) thresholds, discover the complete set of frequent patterns in a database that satisfy both *minsup* and *minRsup* thresholds.

4. PROPOSED APPROACH

In this section, we first discuss the performance problems of RSA algorithm. Subsequently, we introduce the basic idea and proposed RSFP-growth algorithm.

4.1 Performance problems of RSA algorithm

The *relative support* measure can effectively confront the rare item problem while mining frequent patterns involving both frequent and rare items. However, the RSA algorithm has the following issues.

- The RSA is a computationally expensive algorithm because the frequent patterns discovered with the *relative support* measure do not satisfy the *anti-monotic property*. That is, although a pattern satisfies the user-defined *minRsup* threshold, all its non-empty subsets may not have satisfied the *minRsup* threshold.
- It also suffers from the same performance problems as the Apriori algorithm, which includes generating huge number of candidate patterns and multiple scans on the database.

4.2 Basic Idea

The space of items in a database gives rise to a subset lattice. The itemset lattice is a conceptualization of the search space when mining frequent (or user-interest based) patterns. Reducing this search space is an important research issue in pattern mining. The popular techniques to reduce the search space involve the usage of anti-monotonic property, monotonic property, or succinct property (see Definition 1). If a constraint does not satisfy any of these properties, then it is said that mining patterns with it is a computationally expensive process as the mining algorithm has to search the entire lattice space, i.e., 2^I itemsets.

DEFINITION 1. *(Anti-monotone, Monotone and Succinct Constraints.) A constraint* C_a *is anti-monotone if and only if whenever a pattern S violates* C_a*, so does any superset of S. A constraint* C_m *is monotone if and only if whenever a pattern S satisfies* C_m*, so does any superset of S. Succinctness is defined in steps, as follows*

- *A pattern* $X \subseteq I$ *is a succinct set, if it can be expressed as* $\sigma_p(I)$ *for some selection predicate p, where* σ *is the selection operator.*
- $SP \subseteq 2^I$ *is a succinct powerset, if there is a fixed number of succinct sets* $I_1, I_2, \cdots, I_k \subseteq I$*, such that SP can be expressed in terms of the strict powersets of* $I_1, \cdots, I_k$ *using union and minus.*
- *Finally, a constraint* C_s *is succinct provided* $SAT_{C_s}(I)$ *is a succinct powerset.*

Pei et al. [15] have investigated the concept of convertible constraints, and showed that some of the constraints which do not satisfy any of these properties when items in a database are considered as an unordered set can satisfy these properties if items in a database are considered as an ordered set. These properties are known as the *convertible anti-monotonic* and *convertible monotonic* properties. All the constraints or measures discussed in the literature do not satisfy either of these two properties. Therefore, *identifying whether a constraint satisfies the convertible anti-monotonic property, convertible monotonic property, or neither of these two properties is a research problem in pattern mining.*

To reduce the computational cost, we have investigated the nature of *relative support* measure and found that it satisfies the convertible anti-monotonic property if items are arranged in *support* order. The correctness is based on Property 1 and Lemma 1 and is shown in Theorem 1. Unlike the anti-monotonic property, the definition of convertible anti-monotonic property varies with respect to the measure(s) and the order in which items are to be arranged to satisfy this property. Definition 2 defines the convertible anti-monotonic property for the patterns discovered using the *relative support* and *support* measures.

DEFINITION 2. ***(The convertible anti-monotonic property of a frequent pattern.)*** *If a* sorted k-pattern, $\{i_1, i_2, \cdots, i_k\}$, $k \geq 2$ *and* $S(i_1) \geq S(i_2) \geq \cdots \geq S(i_k)$*, is frequent, then all its non-empty subsets containing the item having lowest* support *within it (i.e.,* i_k*) will also be frequent. That is, if* $Rsup(X) \geq minRsup$ *and* $S(X) \geq minsup$*, then* $\forall Y \subseteq X$ *and* $i_k \in Y$*,* $Rsup(Y) \geq minRsup$ *and* $S(Y) \geq minsup$.

EXAMPLE 2. *In a transactional database, let xyz be a sorted frequent 3-pattern such that* $S(x) \geq S(y) \geq S(z)$*. Since* $xz \subset xyz$*, it turns out that xz is a frequent pattern as* $Rsup(xz) \geq Rsup(xyz) \geq minRsup$ *and* $S(xz) \geq S(xyz) \geq minsup$*. Similarly, yz is also a frequent pattern because* $Rsup(yz) \geq Rsup(xyz) \geq minRsup$ *and* $S(yz) \geq S(xyz) \geq minsup$*). The 1-pattern z is also a frequent pattern because* $S(z) \geq S(xyz) \geq minsup$*. Thus, in a sorted frequent 3-pattern xyz, all its non-empty subsets containing the item with lowest frequency within it (i.e., z, xz and yz) are also frequent. Please note that although* $xy \subset xyz$*, we cannot say* $Rsup(xyz)$ *will be less than or equal to* $Rsup(xy)$ *because there exists a case where* $\frac{S(xyz)}{S(z)} \not\leq \frac{S(xy)}{S(y)}$.

Pei et al. [15] have also theoretically shown that the convertible anti-monotonic property is same as the anti-monotonic property for a pattern-growth algorithm. Therefore, initially, we have extended the existing FP-growth [6] (or the *CFG* algorithm in [15]) to discover the complete set of frequent patterns using both *support* and relative *support* measures. It involved the following two steps:

i. Compress the database into the FP-tree, which retains the itemset association information.

ii. Using each interesting item in the FP-tree as an initial **suffix item** (or suffix pattern), construct its conditional pattern base consisting of the set of complete prefix paths in the FP-tree co-occurring with the suffix item, then construct its conditional FP-tree with all those items that have *support* and *relative support* no less than the respective *minsup* and *minRsup* thresholds, and perform mining recursively on such a FP-tree. The pattern-growth is achieved by the concatenation of

the suffix pattern with the interesting patterns generated from the conditional FP-tree. The correctness of the algorithm is shown in Lemma 3.

We have observed that such approach is not an effective way to discover the complete set of frequent patterns as the every item in the conditional pattern base of a suffix pattern has to go through two checks, namely *minsup* and *minRsup*, to derive the corresponding conditional FP-tree. To reduce these number of checks, we redefine the frequent pattern using the notion of sorted set of items in a pattern. Definition 3 provides the alternative definition of a frequent pattern using the *support* and *relative support* measures. The correctness is shown in Lemma 2.

DEFINITION 3. *Given the user-specified minsup and minRsup constraints, a sorted k-pattern X, $S(i_1) \geq S(i_2) \geq \cdots \geq S(i_k)$, is said to be frequent if*

$$S(X) \quad \geq \quad max(S(i_k) \times minRsup,\ minsup). \tag{2}$$

EXAMPLE 3. *Continuing with Example 1, the pattern ab is frequent because*

$$\begin{array}{rcl} S(ab) & \geq & minsup \\ & and & \\ \dfrac{S(ab)}{S(b)\ (=min(S(a),S(b)))} & \geq & minRsup \end{array} \tag{3}$$

From Equation 3, it turns out that for the frequent pattern ab its $S(ab) \geq max(minRsup \times S(b),\ minsup)$.

Using Definition 3 and the prior knowledge regarding the FP-tree construction and mining technique, we introduce a novel concept known as the *conditional minimum support* (*Cminsup*). It is defined in Definition 4, and correctness is shown in Lemma 4.

DEFINITION 4. *(The conditional minimum support of a suffix pattern). Let i_j be the initial suffix item (or 1-pattern) having* support *$S(i_j)$. Let minsup and minRsup be the user-defined minimum* support *and minimum* relative support *thresholds. The conditional minimum* support *of a suffix pattern $\alpha \ni i_j$, denoted as $Cminsup(\alpha)$, is $max(minRsup \times S(i_j),\ minsup)$.*

Using the concept of conditional minimum *support*, we propose a pattern-growth algorithm known as Relative Support Frequent Pattern-growth (RSFP-growth), which is discussed in subsequent subsection.

PROPERTY 1. ***(Apriori property.)*** *If X and Y are the patterns such that $Y \subset X$, then $S(Y) \geq S(X)$.*

LEMMA 1. *Let $X = \{i_1, i_2, \cdots, i_k\}$, $1 \leq k \leq n$, be a pattern such that $S(i_1) \geq S(i_2) \geq \cdots, S(i_k)$. If $Y \subset X$ and $i_k \in Y$, then $Rsup(Y) \geq Rsup(X)$.*

PROOF. The *relative support* of X, i.e., $Rsup(X) = \frac{S(X)}{S(i_k)\ (=min(S(i_j)|\forall i_j \in X)}$. The *relative support* of Y, i.e., $Rsup(Y) = \frac{S(Y)}{S(i_k)\ (=min(S(i_j)|\forall i_j \in X)}$. Since $Y \subset X$, it turns out that $S(Y) \geq S(X)$ (Property 1). Thus, $Rsup(Y) \geq Rsup(X)$ as $\frac{S(Y)}{S(i_k)} \geq \frac{S(X)}{S(i_k)}$. □

THEOREM 1. *If the* relative support *of pattern satisfies the user-defined minRsup threshold, then all its non-subsets containing an item having the lowest* support *within it also satisfy the minRsup threshold.*

PROOF. Let $X = \{i_1, i_2, \cdots, i_k\}$, $1 \leq k \leq n$, be a pattern such that $S(i_1) \geq S(i_2) \geq \cdots \geq S(i_k)$. If $Rsup(X) \geq minRsup$, then

$$\frac{S(X)}{S(i_k)\ (=min(S(i_j)|\forall i_j \in X)} \quad \geq \quad minRsup. \tag{4}$$

If Y is a pattern such that $Y \subset X$ and $i_k \in Y$, then based on Lemma 1 it turns out that $Rsup(Y) \geq Rsup(X) \geq minRsup$. Therefore, if the *relative support* of a pattern satisfies the user-defined *minRsup* threshold, then all its subsets containing an item with lowest *support* within it will also satisfy the *minRsup* threshold. □

LEMMA 2. *Let $X = \{i_1, i_2, \cdots, i_k\}$, $1 \leq k \leq n$, be a pattern such that $S(i_1) \geq S(i_2) \geq \cdots, S(i_k)$. For the user-defined minsup and minRsup constraints, the pattern X can be said frequent if and only if $S(X) \geq max(S(i_k) \times minRsup, minsup)$.*

PROOF. From the definition of frequent pattern given in Equation 1, the pattern X can be said frequent if

$$\begin{array}{rcl} S(X) & \geq & minsup \\ & and & \\ \dfrac{S(X)}{min(S(i_1), S(i_2), \cdots, S(i_k))} & \geq & minRsup. \end{array} \tag{5}$$

Since $min(S(i_1), S(i_2), \cdots, S(i_k)) = S(i_k)$, Equation 5 can be expressed as follows:

$$\begin{array}{rcl} S(X) & \geq & minsup \\ & and & \\ \dfrac{S(X)}{S(i_k)} & \geq & minRsup. \end{array} \tag{6}$$

Thus, X can be a frequent patten if and only if $S(X) \geq max(S(i_k) \times minRsup, minsup)$. □

LEMMA 3. *Let α be a suffix pattern in* FP-tree. *Let $min\_item\_sup(\alpha)$ be the minimum* support *of an item in α, i.e., $min\_item\_sup(\alpha) = min(S(i_j)|\forall i_j \in \alpha)$. Let B be α conditional pattern base, and β be an item in B. Let $S(\beta)$ be the* support *of β in the transactional database. Let $S_B(\beta)$ be the* ***conditional support*** *of β, i.e.,* support *of β in B, respectively. If α is frequent, $S_B(\beta) \geq minsup$ and $\frac{S_B(\beta)}{min\_item\_sup(\alpha)} \geq minRsup$, then the pattern $< \alpha, \beta >$ is also a frequent pattern.*

PROOF. According to the definition of conditional pattern base and FP-tree, each subset in B occurs under the condition of the occurrence of α in the transactional database. If an item β appears in B for n times, it appearers with α in n times. Further, $min\_item\_sup(\alpha) = min\_item\_sup(\alpha \cup \beta)$ as FP-tree is constructed in *support* descending order of items. Thus, from the definition of frequent pattern used in this model, if the $S_B(\beta) \geq minsup$ and $\frac{S_B(\beta)}{min\_item\_sup(\alpha)} \geq minRsup$, the pattern $< \alpha, \beta >$ is therefore a frequent pattern. □

LEMMA 4. *Let α be a suffix pattern in* FP-tree *that has resulted from the initial suffix item i_j. Let the $Cminsup(\alpha)$ be the Cminsup of α. Let B be α conditional pattern base, and β be an item in B. Let $S(\beta)$ and $S_B(\beta)$ be the* support *of β in the transactional database and in B, respectively. If α is frequent and $S_B(\beta) \geq Cminsup$, the pattern $< \alpha, \beta >$ is therefore also frequent.*

PROOF. From the mining procedure of FP-tree, the $min\_item\_sup(\alpha) = S(i_j)$. From Lemma 3, it turns out that if the

$S_B(\beta) \geq minsup$ and $\frac{S_B(\beta)}{S(i_j)} \geq minRsup$, the pattern $< \alpha, \beta >$ is a frequent pattern. In other words, $< \alpha, \beta >$ is a frequent pattern if $S_B(\beta) \geq max(minsup, \text{- } minRsup \times S(i_j))$. From the definition of conditional minimum *support*, it can be said that $< \alpha, \beta >$ is a frequent if $S_B(\beta) \geq Cminsup(\alpha)$ $(= max(minsup,\ minRsup \times S(i_j)))$. □

4.3 Relative Support Frequent Pattern-growth

The RSFP-growth algorithm uses pattern-growth technique and the concept of conditional minimum *support* to discover frequent patterns effectively using the *relative support* and *support* measures. Briefly, the RSFP-growth involves the following two steps:

i. Compress the database into the FP-tree, which retains the itemset association information.

ii. Using each interesting item in the FP-tree as an initial **suffix item** (or suffix pattern), measure the conditional minimum *support* (*Cminsup*) and construct its conditional pattern base consisting of the set of complete prefix paths in the FP-tree co-occurring with the suffix item. Next, construct the conditional FP-tree with all those items that have *support* no less than the *Cminsup* threshold in the conditional pattern base, and perform mining recursively on such a FP-tree using *Cminsup*. The pattern-growth is achieved by the concatenation of the suffix pattern with the interesting patterns generated from the conditional FP-tree.

The working of RSFP-growth is shown in Algorithm 1 and described as follows. The RSFP-growth algorithm accepts transactional database, *minsup* and *minRsup* as its input parameters. An FP-tree [6] is created using *minsup* threshold (line 1 in Algorithm 1). Next, the procedure RSFP-mine_1 is called to discovered frequent patterns from FP-tree. The RSFP-mine_1 procedure selects each item in the FP-tree as an initial suffix item (or pattern) and calculates its *Cminsup* (line 2 in Procedure 2). Next, the conditional pattern base and conditional FP-tree are generated for the suffix item using *Cminsup* (lines 3 to 11 in Procedure 2)). Next, the RSFP-mine_k procedure is called to recursively mine frequent patterns from the conditional FP-tree of suffix item.

We now explain the working of RSFP-growth algorithm using the database shown in Table 1. Let the user-defined *minsup* and *minRsup* thresholds be 3 and 0.65, respectively. Scan the database and measure the *support* of items in a database. Prune the infrequent items (i.e., items having *support* less than *minsup*) and sort the remaining items in the order of descending *support*. This resulting set or *list* is denoted *L*. Thus, we have $L = \{\{a:11\}, \{b:9\}, \{c:9\}, \{d:6\}, \{e:5\}, \{f:5\}\}$.

An FP-tree is constructed as follows. First, create the root of the tree, labeled with "null." Scan database *DB* a second time. The scan of the first transaction, "1:a, b," which contains two items (a and b in L order), leads to the construction of the first branch of the tree with two nodes, $\langle a:1 \rangle$ and $\langle b:1 \rangle$, where a is linked as a child of the root and b is linked to a. The second transaction, "2:a, b, e," would result in a branch where a is linked to root, b is linked to a and e is linked to b. However, this branch would share a common prefix, a and b, with the existing path for "1". Therefore, we instead increment the count of a and b by 1, and create a new node, $\langle e:1 \rangle$, which is linked as the child of $\langle b:2 \rangle$. Similar process is repeated for other transactions in the database. To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links. The FP-tree obtained after scanning all transactions of Table 1 is shown in Figure 1 with the associated node-links.

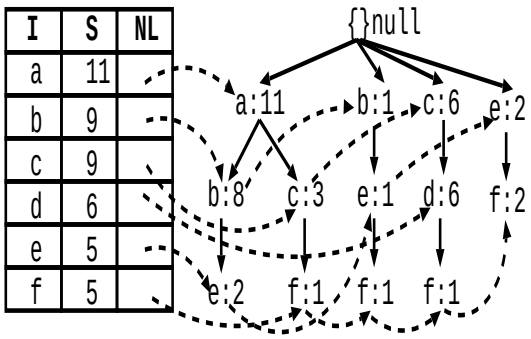

Figure 1: FP-tree. The terms 'I', 'S' and 'NL' respectively denote item, support and node-link.

Table 2: Mining frequent patterns.

SI	*Cminsup*	Conditional Pattern Base	Conditional FP-tree	Frequent patterns
f	3	$\{\{a,c:1\}, \{c,d:1\}, \{e:2\}, \{b,e:1\}\}$	$\langle e:3 \rangle$	{e, f : 3}
e	3	$\{a,b:2\}$	-	-
d	3	$\{c:6\}$	$\langle c:6 \rangle$	{c, d : 6}
c	3	$\{a:3\}$	-	-
b	3	$\{ea:8\}$	$\langle a:8 \rangle$	{a, b : 8}

Mining frequent 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 L, as a suffix item. The item f occurs in four branches of the FP-tree of Figure 1. The *Cminsup* of 'f' is 3 ($\simeq max(3, 5 \times 0.65)$. The paths containing f in FP-tree are $\langle a,c,f:1 \rangle$, $\langle c,d,f:1 \rangle$ $\langle e,f:2 \rangle$ and $\langle b,e,f:1 \rangle$. Therefore, considering f as a suffix, its corresponding four prefix paths are $\langle a,c:1 \rangle$, $\langle c,d:1 \rangle$, $\langle e:2 \rangle$ and $\langle b,e:1 \rangle$, 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 *Cminsup*. The concatenation of suffix pattern with the item in conditional FP-tree generates the frequent pattern $\{e,f:3\}$. Similar process is repeated for the remaining other items in the FP-tree to discover the complete set of frequent patterns.

5. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of FP-growth, RSA and RSFP-growth algorithms. We show that RSFP-growth is a better algorithm to mine frequent 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 memory. The runtime specifies the total execution time, i.e., CPU and I/Os. The runtime is expressed in seconds. We pursued experiments on synthetic (T10I4D100K) and real-world (BMS-WebView-1, Mushroom and Kosarak) datasets. The datasets are available at Frequent Itemset Mining repository [1]. The details of these datasets are shown in Table 3.

5.1 Generation of Frequent Patterns

Figure 2(a), (b) and (c) respectively show the number of frequent patterns generated in *T10I4D100k*, *BMS-WebView-1* and *Mushroom* datasets with the basic model (denoted as *BM*) [2] and the

Table 3: Dataset Characteristics. The terms “Max.”, “Avg.” and “Tran.” are respectively used as the acronyms for “maximum”, “average” and “transaction.“

Dataset	Transa-ctions	Distinct Items	Max. Trans. Size	Avg. Trans. Size	Type
*T*10*I*4*D*100*k*	100000	870	29	10.1	sparse
BMS-WebView-1	59602	4971	267	2.5	sparse
Mushroom	8124	119	23	23.0	dense
Kosarak	990002	41270	2498	8.1	sparse

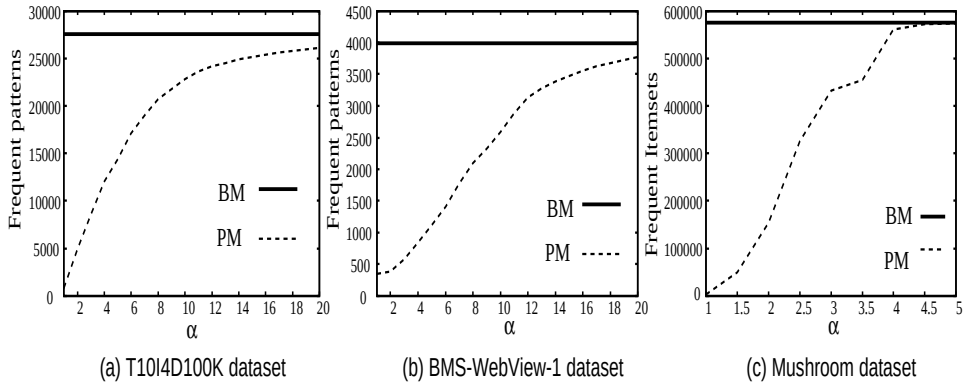

Figure 2: Frequent patterns generated in different databases at different *minRsup* values.

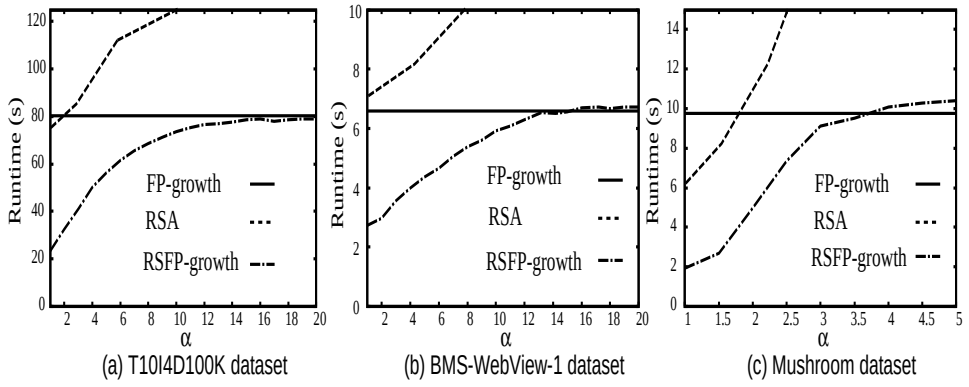

Figure 3: Runtime consumed by different algorithms in different databases at different *minRsup* values.

Algorithm 1 RSFP-growthAlgorithm (*DB*: database, *minsup*: minimum support, *minRsup*: minimum relative support)

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 RSFP-mine_1($Tree, null$).

proposed model (denoted as *PM*) of frequent patterns. The *minsup* thresholds set in $T10I4D100k$, *BMS-WebView-1* and *Mushroom* are 0.1%, 0.1% and 25%, respectively. The *minRsup* threshold in each database is set as $\frac{1}{\alpha}$ and varied α from 1 to 20. The thick lines shows the number of frequent patterns discovered by FP-growth (or basic model). It can be observed that increase in α value has increased the number of frequent patterns in the proposed model. The reason is due to the decrease in *minRsup* threshold with the increase in α value. If $minRsup = 0$ (or α is set too large), then both the models will generate same number of frequent patterns.

5.2 Runtime Comparison of FP-growth, RSA and RSFP-growth Algorithms

Figure 3(a), (b) and (c) respectively show the runtime taken by FP-growth, RSA and FP-growth algorithms in $T10I4D100k$, *BMS-WebView-1* and *Mushroom* datasets. The *minsup* thresholds set in $T10I4D100k$, *BMS-WebView-1* and *Mushroom* are 0.1%, 0.1% and 25%, respectively. The *minRsup* threshold in each database is set as $\frac{1}{\alpha}$ and varied α from 1 to 20. To compare RSA and RFP-growth algorithms, the s_1 and s_2 thresholds of *RSA* algorithm are respectively set to 100% and *minsup* of the database. The following observations can be drawn from these figures.

- Since the number of frequent patterns getting generated with FP-growth remained constant, the runtime of FP-growth has resulted in a straight line.
- Increase in α value has increased the runtime of both RSA and RSFP-growth algorithms. The reason is due to the increase of number of frequent patterns with increase in α value.
- The RSFP-growth algorithm has taken relatively less runtime than the FP-growth algorithm at lower α (i.e., higher *minRsup* values). It is because of the less number of frequent patterns discovered by RSFP-growth.
- At higher α (i.e., at low *minRsup*), the FP-growth and RSFP-growth algorithms have discovered almost same the number of frequent patterns. However, the runtime of RSFP-growth is slightly more than the FP-growth. It is because of the additional runtime was spent by RSFP-growth algorithm in calculating the *conditional minimum support* for each suffix item.

Procedure 2 RSFP-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:    Calculate Cminsup = max(minsup, S(a_i) × minRsup).
3:    Generate pattern β = α ∪ a_i with support = a_i.support.
      {The term support represent support count.} {S(β) = S(a_i)
      in α-projected database}
4:    Get a set I_β of items to be included in β-projected database.
5:    for each item in I_β, compute its support in β-projected
      database;
6:    for each b_j ∈ I_β do
7:       if S(βb_j) < Cminsup then
8:          delete b_j from I_β; {pruning based on conditional mini-
            mum support}
9:       end if
10:   end for
11:   construct β-conditional FP-tree with items in I_β Tree_β.
12:   if Tree_β ≠ ∅ then
13:      RSFP-mine_k(Tree_β, Cminsup, β);
14:   end if
15: end for
```

Procedure 3 RSFP-mine_k($Tree, Cminsup, \alpha$); Constructing the conditional pattern base for length-k, $k > 1$, suffix pattern.

```
1: for each a_i in the header of Tree do
2:    Generate pattern β = α ∪ a_i with support = a_i.support.
3:    Get a set I_β of items to be included in β-projected database.
4:    for each item in I_β, compute its support in β-projected
      database;
5:    for each b_j ∈ I_β do
6:       if S(βb_j) < Cminsup then
7:          delete b_j from I_β; {pruning based on minimum sup-
            port}
8:       end if
9:    end for
10:   construct β-conditional FP-tree with items in I_β Tree_β.
11:   if Tree_β ≠ ∅ then
12:      RSFP-mine_k(Tree_β, Cminsup, β);
13:   end if
14: end for
```

Please note that the runtime of RSFP-growth can be much higher than FP-growth if *conditional minimum support* is not used.

- At any α value (i.e., irrespective of *minRsup* threshold), RSFP-growth is better than the RSA algorithm. It is because of two reasons: first, the search space of RSA was more than the search space of RSFP-growth algorithm; second, RSA suffered from the same performance problems as the Apriori algorithm.

5.3 Scalability Test on RSA and RSFP-growth Algorithms

In this experiment, we evaluate the scalability performance of RSA and RSFP-growth algorithms on 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 five portions of 0.2 million transactions in each part. Then we investigated the performance of RSA and RSFP-growth algorithms after accumulating

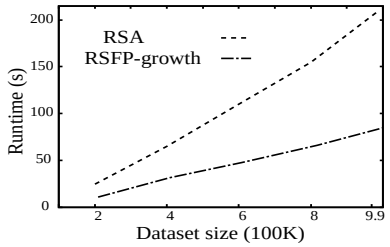

Figure 4: Scalability of RSA and RSFP-growth algorithms.

each portion with previous parts while performing correlated pattern mining each time. We fixed $minsup = 0.1\%$ and $minRsup = 0.5$ (i.e., $\alpha = 2$) for each experiment. The experimental results are shown in Figure 4. The runtime in *y*-axes of Figure 4 specify the total runtime consumed by RSA and RSFP-growth algorithms with the increase of database size. It is clear from the graphs that as the database size increases, overall runtime increases for both RSA and RSFP-growth algorithms. However, the RSFP-growth algorithm requires relatively less runtime than RSA algorithm. Therefore, it can be observed from the scalability test that RSFP-growth can efficiently mine frequent patterns over large datasets and distinct items with considerable amount of runtime.

Overall, the RSFP-growth algorithm is runtime efficient and scalable than the RSA algorithm.

6. CONCLUSIONS AND FUTURE WORK

This paper has proposed an efficient and effective pattern-growth algorithm to discover the complete set of frequent patterns using *relative support* and *support* measures. The paper has also shown that it is not computationally expensive to mine the patterns as the *relative support* measure satisfies the *convertible anti-monotonic property* if items within a pattern are arranged in *support* order. A novel concept known as conditional minimum *support* has been introduced and extended to FP-growth algorithm to discover frequent patterns. By conducting experiments on various datasets, we have shown that RSFP-growth outperforms RSA algorithm with respect to both runtime and scalability.

The future works of the paper are as follows: first, data mining techniques, such as classification and clustering, employ frequent patterns discovered with single *minsup* constraint to improve their performance. As a result, these techniques also suffer from the rare item problem. It is interesting to investigate the usage of frequent patterns discovered with the relative support measure to address the problem. Second, the interestingness of frequent patterns discovered using various measures, such as *relative support* and *all-confidence*, needs to be investigated.

7. 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: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, New York, NY, USA, 1993. ACM.

[3] 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.

[4] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: generalizing association rules to correlations. *SIGMOD Rec.*, 26(2):265–276, 1997.

[5] F. Gedikli and D. Jannach. Neighborhood-restricted mining and weighted application of association rules for recommenders. In *Proceedings of the 11th international conference on Web information systems engineering*, WISE'10, pages 157–165, Berlin, Heidelberg, 2010. Springer-Verlag.

[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] R. U. Kiran and P. K. Reddy. An improved frequent pattern-growth approach to discover rare association rules. In *KDIR*, pages 43–52, 2009.

[8] R. U. Kiran and P. K. Reddy. Improved approaches to mine rare association rules in transactional databases. In *IDAR '10: Proceedings of the Fourth SIGMOD PhD Workshop on Innovative Database Research*, pages 19–24, New York, NY, USA, 2010. ACM.

[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] R. U. Kiran and P. K. Reddy. Novel techniques to reduce search space in multiple minimum supports-based frequent pattern mining algorithms. In *EDBT*, pages 11–20, 2011.

[11] M. Kubat, R. C. Holte, and S. Matwin. Machine learning for the detection of oil spills in satellite radar images. *Mach. Learn.*, 30(2-3):195–215, Feb. 1998.

[12] 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, New York, NY, USA, 1999. ACM.

[13] E. R. Omiecinski. Alternative interest measures for mining associations in databases. *IEEE Trans. on Knowl. and Data Eng.*, 15(1):57–69, 2003.

[14] J. Pei and J. Han. Constrained frequent pattern mining: a pattern-growth view. *SIGKDD Explor. Newsl.*, 4(1):31–39, 2002.

[15] J. Pei, J. Han, and L. V. Lakshmanan. Pushing convertible constraints in frequent itemset mining. *Data Mining and Knowledge Discovery*, 8:227–252, 2004. 10.1023/B:DAMI.0000023674.74932.4c.

[16] 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.

[17] A. Surana, R. U. Kiran, and P. K. Reddy. Selecting a right interestingness measure for rare association rules. In *COMAD*, page 115, 2010.

[18] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 32–41, New York, NY, USA, 2002. ACM.

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

[20] 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.

[21] 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.