On Computing Condensed Frequent Pattern Bases

Jian Pei
State Univ. of New York at Buffalo
jianpei@cse.buffalo.edu

Guozhu Dong
Wright State Univ.
gdong@cs.wright.edu

Wei Zou
Jiangxi Normal Univ.
zouwei@jxnu.edu.cn

Jiawei Han
Univ. of Illinois
hanj@cs.uiuc.edu

Abstract

Frequent pattern mining has been studied extensively. However, the effectiveness and efficiency of this mining is often limited, since the number of frequent patterns generated is often too large. In many applications it is sufficient to generate and examine only frequent patterns with support frequency in close-enough approximation instead of in full precision. Such a compact but close-enough frequent pattern base is called a **condensed frequent patterns-base**.

In this paper, we propose and examine several alternatives at the design, representation, and implementation of such condensed frequent pattern-bases. A few algorithms for computing such pattern-bases are proposed. Their effectiveness at pattern compression and their efficient computation methods are investigated. A systematic performance study is conducted on different kinds of databases, which demonstrates the effectiveness and efficiency of our approach at handling frequent pattern mining in large databases.

1 Introduction

It has been well recognized that frequent pattern mining plays an essential role in many important data mining tasks (e.g., mining association [2]). However, it has also been widely recognized that frequent pattern mining often produces a huge number of patterns [15], which reduces not only the efficiency but also the effectiveness of mining, since it is unrealistic to store and comprehend so many patterns. Recently, efforts have been devoted to address this problem. In general, interesting proposals can be classified into two categories. First, concise representations of frequent patterns have been explored, such as *frequent closed patterns* [12, 15, 14], that can be used to remove sub-patterns which have the same support as some of their super-patterns. Studies like [15] have shown that, by doing so, the total number of patterns and rules can be reduced substantially, especially in dense data sets. Second, constraints can be used to capture users' focus, and effective strategies have been developed to push various constraints deep into the mining process [11, 9, 13].

Even though these approaches are useful, they may not be powerful enough in many cases. The compression by the closed-pattern approach may not be so effective since there often exist slightly different counts between super- and sub- patterns. Constraint-based mining, though useful, can hardly be used for pre-computation since different users may likely have different constraints.

Although it seems to be inherent that a large database may contain numerous frequent patterns, it is easy to observe a simple fact in practice: "*Most applications will not need* **precise** *support information of frequent patterns, a* **good approximation** *on support count could be more than adequate.*" Here, by "*good approximation*", we mean that the frequency of every frequent pattern can be estimated with a *guaranteed maximal error bound*. For example, for a frequent pattern $\{$diaper, beer$\}$, instead of giving the exact support count (e.g., 10000), a range, e.g., $10000 \pm 1\%$, may be good enough; the range is a user-specified *error bound*.

"*Why is a condensed frequent pattern base acceptable and often more preferable?*" First, *when mining large database, a small deviation often has a very minor effect on analysis*. For an analyst, the exact information "*diaper and beer have been bought together* $10,050$ *times out of the* 10 *million transactions*" and an approximation "*diaper and beer have been bought together* $10,050 \pm 50$ *times*" may not have any essential difference. Analysis often has to deal with approximation sooner or later, by truncation or rounding. What an analyst is really concerned is that a specified error bound is guaranteed.

Second, *condensing frequent pattern base leads to more effective frequent pattern mining*. By computing a condensed pattern base, the number of patterns can be reduced dramatically, but the general information about frequent patterns still retain. A much smaller base of patterns certainly helps users comprehend the mining results.

Third, *computing a condensed frequent pattern base may lead to more efficient frequent pattern mining*. A condensed frequent pattern base could be much smaller than the complete frequent pattern base. Thus, one may need to compute and access a much smaller pattern base, which leads to better efficiency.

In summary, mining a condensed frequent pattern base may make frequent pattern mining more realistic in real-life applications. In this paper, we introduce the concept of *condensed frequent pattern-base with guaranteed maximal error bound* and study the *efficient computation of such a condensed pattern-base*, with the following contributions. First, we introduce the concept *condensed frequent pattern-base* and devise systematic representations of such frequent pattern-bases. We show that such representations achieve satisfactory approximation with a guaranteed maximal error bound on the support. Second, we develop efficient algorithms for computing condensed pattern bases from transaction databases directly. Our algorithms facili-

tate the relaxation of counting requirement and prune many patterns in the mining. Third, we present a systematic performance study to verify the effectiveness and efficiency of condensed frequent pattern bases. Our results show that computing condensed frequent pattern base is promising. Previously, the ideas of approximating frequent patterns have been probed in some related studies. For example, [10] shows that approximative association rules are interesting and useful. In [4], the notion of free-sets is proposed and can lead to an error-bound approximation of frequencies. However, none of the previous studies systematically explores the problem of designing and mining condensed frequent pattern-based with guaranteed maximal error bound.

The remaining of this paper is organized as follows. The problem of computing a condensed frequent pattern base is introduced in Section 2. A level-by-level frequent pattern base construction method is presented in Section 3. In Section 4, we develop an effective and efficient method for frequent pattern-base construction using max-patterns at various layers. Section 5 presents a comprehensive performance study to demonstrate the effectiveness and efficiency of our approach. Section 6 concludes the study.

2 Problem Definition

We first review some standard terminology for frequent pattern mining. Let $I = \{i_1, \ldots, i_n\}$ be a set of literals, called *items*. An *itemset* (or *pattern*) X, denoted as $X = i_{j_1} \cdots i_{j_l}$ (i.e., by omitting set brackets), is a subset of items in I. An itemset with l items is called an *l-itemset*. For two patterns X and Y such that $X \subseteq Y$, Y is called a *super-pattern* of X, and X a *sub-pattern* of Y.

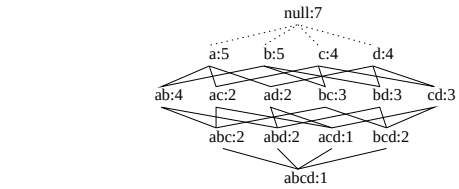
A *transaction* $T = (tid, X)$ is a tuple where tid is a *transaction-id* and X is an itemset. A transaction $T = (tid, X)$ is said to *contain* itemset Y if $Y \subseteq X$. A *transaction database* TDB is a set of transactions. The *support* of an itemset X in TDB, denoted as $sup(X)$, is the number of transactions in TDB containing X, i.e., $sup(X) = |\{(tid, Y)|((tid, Y) \in TDB) \wedge (X \subseteq Y)\}|$.

Given a transaction database TDB and a *support threshold* min_sup, an itemset X is called a *frequent itemset* or a *frequent pattern* if $sup(X) \geq min\_sup$. The problem of frequent pattern mining is to find the complete set of frequent patterns from TDB w.r.t. a user-specified support threshold min_sup. The set of all frequent patterns is called a *frequent pattern base*, or *FP-base* in short.

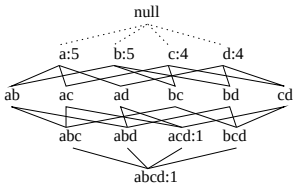
It is often expensive to find the complete set of frequent patterns, since an FP-base may contain a huge number of frequent patterns. In this paper, we propose to overcome the difficulty caused by "huge amount of frequent patterns" as follows: we compute a smaller set of frequent patterns, i.e., a "condensed FP-base", and then use it to approximate the supports of arbitrary frequent patterns.

Problem statement. Given a transaction database, a support threshold, and a user-specified *error bound* k, the *problem of computing a condensed FP-base* is to find a subset of frequent patterns $\mathcal{B}$ and a function $f_{\mathcal{B}}$ such that the following holds for each pattern X:

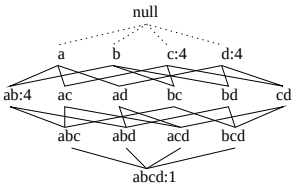
$$f_{\mathcal{B}}(X) = \begin{cases} 0 & \text{if } X \text{ is infrequent} \\ [sup_{lb}, sup_{ub}] \text{ s.t. } (sup_{lb} \leq sup(X) \leq sup_{ub}) & \\ \quad \text{and } (sup_{ub} - sup_{lb}) \leq k & \text{if } X \text{ is frequent} \end{cases}$$



(a) Lattice of frequent patterns



(b) Condensed FP-base Bd



(c) Condensed FP-base Bm

Figure 1. Lattice of frequent patterns for Example 1.

The function $f_{\mathcal{B}}$ is called a *(support) approximation function*, and the set $\mathcal{B}$ a *condensed FP-base* w.r.t. $f_{\mathcal{B}}$.[1] ■

Example 1 Consider the transaction database shown in Table 1. Let the support threshold be $min\_sup = 1$ and the error bound be $k = 2$. The lattice of totally 15 frequent patterns is shown in Figure 1(a).

Transaction-id	Itemset
10	a
20	ab
30	abc
40	$abcd$
50	cd
60	abd
70	bcd

Table 1. A transaction database with seven transactions.

The set $\mathcal{B}_d = \{a : 5, b : 5, c : 4, d : 4, acd : 1, abcd : 1\}$ is a *condensed FP-base*. Patterns in $\mathcal{B}_d$ are those labelled with supports in Figure 1(b). For each pattern X, the function $f_{\mathcal{B}_d}$ is defined as follows:

$$f_{\mathcal{B}_d}(X) = \begin{cases} 0 & \text{if there exists no } X' \in \mathcal{B}_d \text{ s.t. } X \subseteq X' \\ [sup(X), sup(X)] & \text{if } X \in \mathcal{B}_d \\ [sup(X_0) - 2, sup(X_0)], \text{ where } (X_0 \subset X \text{ and} & \\ \quad sup(X_0) = \min(sup(X'')) \text{ for } X'' \subset X & \\ \quad \text{and } X'' \in \mathcal{B}_d), & \text{otherwise} \end{cases}$$

For example, $f_{\mathcal{B}_d}(abcde) = 0$ for the infrequent pattern $abcde$, since there is no $X' \in \mathcal{B}_d$ s.t. $abcde \subseteq X'$; $f_{\mathcal{B}_d}(ac) = [4-2, 4] = [2, 4]$, since c is a sub-pattern of ac in $\mathcal{B}_d$ with the smallest support count (of 4). Here, we used the well known "*Apriori*" property that $supp(X) \geq supp(Y)$ if $X \subseteq Y$. One can verify that, $f_{\mathcal{B}_d}$ can approximate the support count of each frequent pattern as required by the definition given above. For example, $sup(ab)$ is approximated by $[3, 5]$ and $sup(abc)$ is by $[2, 4]$.

[1] Instead of an absolute error bound k, a relative, percentage-based error bound $k\%$ can also be used to compute a condensed FP-base. In this case, $\frac{sup_{ub} - sup_{lb}}{sup_{lb}} \leq k\%$ should be satisfied for frequent patterns.

Moreover, $\mathcal{B}_m = \{c : 4, d : 4, ab : 4, abcd : 1\}$ is another condensed FP-base, as plotted in Figure 1(c). The corresponding approximation function $f_{\mathcal{B}_m}$ is defined (for each pattern X) as follows:

$$f_{\mathcal{B}_m}(X) = \begin{cases} 0 & \text{if there exists no } X' \in \mathcal{B}_m \text{ s.t. } X \subseteq X' \\ [sup(X), sup(X)] & \text{if } X \in \mathcal{B}_m \\ [sup(X_0), sup(X_0) + 2] & \text{where } (X_0 \supset X \\ & \text{and } sup(X_0) = \max\{sup(X'') \mid \\ & X'' \supset X, X'' \in \mathcal{B}_m\}), \quad \text{otherwise} \end{cases}$$

Condensed FP-bases and approximation functions are not unique. A superset of a condensed FP-base is also a base w.r.t. the identical approximation function. A condensed FP-base is *minimal* (w.r.t. an approximation function f) if it does not contain a proper subset which is also a condensed FP-base w.r.t. f. Interestingly, even minimal condensed FP-bases are not unique. For Example 1, both $\mathcal{B}_d$ and $\mathcal{B}_m$ are *minimal* condensed FP-bases.

Among possible approximation bases, we prefer those requiring as little space as possible. Such condensed FP-bases offer significant compression effect, which can be measured by *compression ratio* δ, defined as

$$\delta = \frac{\text{\# of patterns in the condensed FP-base}}{\text{total \# of frequent patterns}} \qquad (1)$$

Clearly, the smaller the compression ratio, the better the compression effect. We observe from Example 1 that condensed FP-bases can produce considerable space savings even with a small error bound. For example, $\mathcal{B}_d$ achieves a compression ratio of 40%, whereas $\mathcal{B}_m$ achieves 26.7%. $\mathcal{B}_m$ achieves better compression than $\mathcal{B}_d$.

Previous research also considered computing reduced sets of frequent patterns, including reduction based on frequent closed itemsets [12] and containment based reduction [3, 6]. An itemset X is called a *closed pattern* if there exists no proper superset X' of X such that $sup(X) = sup(X')$, while X is called a *max-pattern* if there exists no superset X' of X such that X' is also frequent. Interestingly, it can be shown that the complete set of frequent closed patterns is a minimal condensed FP-base with error bound 0, while the complete set of max-patterns is a minimal condensed FP-base with error bound $(|TDB| - min\_sup)$, where min_sup is the support threshold. However, none of these considers approximating supports of frequent patterns with a user-specified error bound as we do here.

How can we construct condensed FP-bases effectively and efficiently? This is the topic of the following sections.

3 Constructing a Condensed FP-Base Level-by-level

We now consider an approach that constructs a condensed FP-base by examining all frequent patterns level-by-level: A frequent pattern is added into the condensed FP-base only if it cannot be approximated by its sub-patterns in the base. The method is illustrated next.

Example 2 A condensed FP-base $\mathcal{B}_d$ for the transaction database TDB in Table 1, for the support threshold of 1 and the error bound of 2 is constructed as follows (as shown in Figure 1(b)), where the approximation function $f_{\mathcal{B}_d}$ is defined in Example 1.

For each pattern X, let $X.ub$ denote $\min\{sup(X') | X' \in \mathcal{B}$, and $X' \subseteq X\}$, i.e., $X.ub$ is the minimum of supports of all sub-patterns of X currently in $\mathcal{B}_d$.

Step 1. We initialize $\mathcal{B}_d = \emptyset$ and mine length-1 and length-2 frequent patterns. Since length-1 frequent patterns are the "*most frequent-end borders*" of frequent patterns and none of their sub-patterns is in the base, we insert all of them (i.e. a, b, c, and d) into $\mathcal{B}_d$.

For each length-1 frequent pattern x, $x.ub = sup(x)$.

Step 2. For the next level, i.e., length-2 frequent patterns, we have $xy.ub = \min(x.ub, y.ub)$ for each length-2 frequent pattern xy.

We do two types of insertions into $\mathcal{B}_d$.

(i) A length-2 frequent pattern X is added into $\mathcal{B}_d$ if $X.ub - sup(X)$ is over the error bound (i.e., if its support cannot be approximated by its sub-patterns in the base $\mathcal{B}_d$). In this example, since all length-2 frequent patterns can be approximated properly by their length-1 sub-patterns, no length-2 frequent pattern is inserted into $\mathcal{B}_d$.

(ii) If a length-2 frequent pattern has no frequent length-3 super-pattern, i.e., it is a max-pattern, then it is inserted into $\mathcal{B}_d$. Max-patterns are needed in $\mathcal{B}_d$ since they are used to determine whether a pattern is frequent. In this example, no such length-2 frequent pattern exists.

Step 3. For the length-3 level, since $acd.ub - sup(acd) = 4 - 1 > 2$, pattern acd is inserted into $\mathcal{B}_d$. Here, $abc.ub = \min(ab.ub, ac.ub, bc.ub)$. After the insertion, we set $abc.ub = sup(abc) = 1$. We then mine length-4 frequent patterns and see that there is no length-3 max-pattern.

Step 4. The length-4 frequent pattern $abcd$ is a max-pattern (since there is no length-5 frequent pattern), and it is inserted into $\mathcal{B}_d$.

At the end, the base $\mathcal{B}_d$ contains 6 patterns: a, b, c, d, acd, and $abcd$. Since the search is downward from length-1 patterns, we call the resulting base as a *downward condensed FP-base*. ∎

Let us generalize the level-by-level condensed FP-base construction method. We first define the approximation function ϑ as follows.

Definition 1 Given a condensed FP-base $\mathcal{B}$, an error bound k and a pattern X.

$$\vartheta(X) = \begin{cases} 0 & \text{if there exists no } X' \in \mathcal{B} \\ & \text{s.t. } X' \supseteq X \\ [sup(X), sup(X)] & \text{if } X \in \mathcal{B} \\ [m - k, m] & \text{if } X \notin \mathcal{B}, \text{ where } m = \\ & \min\{sup(X') \mid X' \in \mathcal{B}, \\ & X' \subset X\} \end{cases}$$

∎

The following algorithm computes a condensed FP-base with respect to approximation function ϑ.

Algorithm 1 *(*CFP-D*: a Level-by-level downward search method)*

Input: transaction database TDB, support threshold min_sup, and error bound k;

Output: a condensed FP-base $\mathcal{B}$ w.r.t. ϑ;

Method:

1. let $\mathcal{B} = \emptyset$;
2. find length-1 frequent patterns and insert them into $\mathcal{B}$; for each length-1 frequent pattern X, let $X.ub = sup(X)$;
3. let $i = 2$;
4. generate the set $\mathcal{F}_i$ of length-i frequent patterns; for each length-i frequent pattern X, let $X.ub = \min(X'.ub)$, where X' ranges over length-$(i-1)$ sub-patterns of X;
   /* the calculation of $X.ub$ can be done as a byproduct of candidate-generation */
5. if $(X.ub - sup(X)) > k$, then insert X into $\mathcal{B}$ and set $X.ub = sup(X)$;
6. for each length-$(i-1)$ frequent pattern X s.t. X has no super-pattern in $\mathcal{F}_i$, insert X into $\mathcal{B}$;
   /* rationale: X is a max-pattern */
7. if $\mathcal{F}_i \neq \emptyset$ then let $i = i + 1$ and goto Step 4;
8. return $\mathcal{B}$. ∎

One advantage of the method shown in Example 2 is that it is intuitive and can be easily integrated into the Apriori algorithm. The correctness and effectiveness of the algorithm are obvious. Limited by space, we omit the proof here.

What kind of patterns are included in $\mathcal{B}$ computed by Algorithm 1? A frequent pattern X is called a *seed pattern* if for each proper sub-pattern $X' \subset X$, $sup(X') > sup(X)$. Interestingly, it is easy to show that *every pattern in $\mathcal{B}$ computed by Algorithm 1 is either a seed pattern or a max-pattern.*

4 Constructing a Condensed FP-base Using Max-patterns

While Algorithm 1 is intuitive and correct, it has to check every frequent pattern. When there are many frequent patterns, the mining cost is non-trivial. *Can we avoid checking every frequent pattern when constructing a condensed FP-base?* In this section we will answer this question positively by providing a type of condensed FP-base and efficient mining techniques to find such a base.

Intuitively, we are going to construct a condensed FP-base consisting of maximal frequent patterns for a series of support thresholds. More specifically, given a support threshold min_sup and error bound k, we divide the set of frequent patterns into a number of disjoint subsets: (1) the set of patterns with support in the range $[min\_sup, min\_sup+k]$, (2) those with support in the range $[min\_sup + k + 1, min\_sup + 2k + 1]$, etc. The i-th subset contains those patterns with support in the range

$[min\_sup + (i-1)(k+1), min\_sup + ik + i - 1]$ where $(1 \leq i \leq \frac{|TDB|+1-min\_sup}{k+1})$.

Given a frequent pattern, we can approximate its support with maximal error of k, by determining which subset the pattern belongs to. To determine which subset a pattern belongs to, we only need to record the max-patterns at various layers w.r.t. the lower bounds of supports of the ranges. The idea is illustrated in the following example.

Example 3 Given the transaction database TDB in Table 1, support threshold of 1 and error bound of 2, a condensed FP-base $\mathcal{B}_m$ can be constructed as follows.

Since the support threshold is 1 and the total number of transactions in the database is 7, we consider three ranges of supports: $[1, 3]$, $[4, 6]$, and $[7, 7]$. We mine max-patterns w.r.t. support threshold 1, 4, and 7, respectively. The only max-pattern w.r.t. support threshold 1 is $abcd$, the max-patterns w.r.t. support threshold 4 are ab, c and d, while there is no max-pattern w.r.t. support threshold 7. These four patterns form a condensed FP-base $\mathcal{B}_m$.

The base $\mathcal{B}_m$ is shown in Figure 1(c). The approximation function is $f_{\mathcal{B}_m}$, as defined in Example 1. In essence, for each given pattern X we find the super-pattern Y of X in $\mathcal{B}_m$ having the largest support, and use the range of the support for Y as the estimate of the support of X. ∎

We now generalize the ideas by providing the definition of a condensed FP-base.

Definition 2 Given a transaction database TDB, support threshold min_sup and error bound k, let the *number of levels* be

$$n\_level = \lfloor \frac{|TDB| + 1 - min\_sup}{k+1} \rfloor$$

Define

$$min\_sup_1 = min\_sup$$

$$min\_sup_2 = min\_sup + k + 1$$

$$\cdots$$

$$min\_sup_i = min\_sup + (i-1)(k+1) \text{ for } (1 \leq i \leq n\_level)$$

Then, $\mathcal{B} = \bigcup_{i=0}^{i<n\_level} \mathcal{M}_i$ is called an *M-base*, w.r.t. the approximation function ζ defined below. Here $\mathcal{M}_i$ is the set of max-patterns w.r.t. support threshold min_sup_i.

The name *M-base* is used because the base is based on max-patterns.

Definition 3 Given an error bound k, an M-base $\mathcal{B}$, and a pattern X, let

$$\zeta(X) = \begin{cases} 0 & \text{if there exists no } X' \in \mathcal{B} \text{ s.t. } X' \supseteq X \\ [sup(X), sup(X)] & \text{if } X \in \mathcal{B} \\ [m, m+k] & \text{if } X \notin \mathcal{B}, \text{ where } m = \max\{sup(X') \mid X' \in \mathcal{B}, X' \supset X\} \end{cases}$$

It can be shown that each M-base is not only a proper condensed FP-base w.r.t. function ζ but also a minimal one. Limited by space, we omit the formal result here.

The remaining problem is *how to find the max-patterns efficiently in the condensed FP-base $\mathcal{B}_m$.*

There are many methods for mining max-patterns, such as MaxMiner [3], Depth-first Search [1], MAFIA [5], and GenMax[7]. A naïve method to compute $\mathcal{B}_m$ is to call a max-pattern mining algorithm multiple times, once for each lower bound of the ranges as a support threshold.

How do we mine the patterns of M-bases more efficiently than the naïve method? Roughly speaking, we will propose an algorithm to mine the database only once, for all the max-patterns w.r.t. the series of support thresholds. The algorithm proceeds in a depth-first manner. Moreover, our algorithm also uses additional pruning techniques. We will demonstrate the spirit of our algorithm with the following example.

Example 4 Consider the mining of max-patterns w.r.t. support thresholds of 1 and 4 in the M-base $\mathcal{B}_m$ for the transaction database TDB of Table 1.

By scanning the transaction database TDB once, all frequent items, namely $a : 5$, $b : 5$, $c : 4$, and $d : 4$, are found. These items are sorted in support descending order, producing the list *F-list*$= a - b - c - d$.

F-list can be used to divide all max-patterns into four disjoint subsets: (1) the set of max-patterns containing item a; (2) those containing item b but no a; (3) those containing item c but no a nor b; and (4) those containing item d, i.e., the pattern d itself, if it is a max-pattern. We mine these four subsets of max-patterns one by one.

1. To find max-patterns containing item a, we form the *a-projected database* TDB_a by collecting all transactions containing item a, namely b, bc, bcd, and bd.

Items b, c, and d are local frequent items in TDB_a. A list *F-list*$_a = b - c - d$ is formed by sorting these local frequent items in local support descending order. Based on *F-list*$_a$, all max-patterns containing item a can be further divided into four disjoint subsets: (1) pattern a itself, if it is a max-pattern; (2) those containing ab; (3) those containing item ac but no b; and (4) pattern ad if it is a max-pattern. We mine them one by one recursively.

1(a). The support of b in TDB_a is 4, denoted as $sup_{TDB_a}(b) = 4$. Since $sup(ab) = sup_{TDB_a}(b) = 4$, pattern a is not a max-pattern w.r.t. support threshold 4.

1(b). To find max-patterns containing ab, we form the *ab-projected database* TDB_{ab}, which contains c, cd, and d. Items a and b are omitted in TDB_{ab}, since they appear in every transaction in the ab-projected database. There is no item having support 4 or over in TDB_{ab}. Thus, ab is a max-pattern w.r.t. support threshold 4 (the lower bound of the second range of supports).

Items c and d are frequent in TDB_{ab}. We recursively mine max-patterns by forming projected databases. It can be checked that $abcd$ is a max-pattern w.r.t. support threshold 1. Thus, the max-patterns containing ab are $ab : 4$ itself and $abcd : 1$.

1(c). To find max-patterns containing ac but not b, we form ac-projected database TDB_{ac}, which contains d. Here, items a, b and c are omitted since ac appears in every transaction and b occurs before c in F-list. The only frequent item in TDB_{ac} is d. However, $ac \subset abcd$ and $4 > sup(ac) > sup(abcd) = 1$. That means there exists no max-pattern containing ac but no b.

1(d). Since $4 > sup(ad) > sup(abcd) = 1$, ad is not a max-pattern.

Therefore, the max-patterns containing a are ab and $abcd$.

2. To find all max-patterns containing b but not a, we form the b-projected database, which contains c, cd, d, and cd. The local frequent items in TDB_b are c and d, and *F-list*$_b = c - d$. The max-patterns containing b but not a can be divided into three subsets: (1) pattern b itself, if it is a max-pattern; (2) those containing bc; and (3) pattern bd, if it is a max-pattern. Let us mine them one by one.

Since $b \subset ab$ and ab is a max-pattern, b is not a max-pattern;

Since $sup(bc) = sup_{TDB_b}(c) = 2$, we have $4 > sup(bc) \geq sup(bcd) > sup(abcd)$. It follows that there are no max-pattern containing bc but not a.

Similarly, we can check that bd is not a max-pattern.

Thus, there are no max-patterns containing b but not a.

3. To mine max-patterns containing c but not a nor b, we can form the c-projected database and mine it recursively. It can be verified that c is the only such max-pattern.

4. similarly, it can be verified that d is a max-pattern.

Thus the complete set of max-patterns for condensed FP-base $\mathcal{B}_m = \{ab : 4, abcd : 1, c : 4, d : 4\}$. ∎

As shown in the example, the general framework is the depth-first search. A list of frequent items in support descending order, called *F-list*, is used to divide the data as well as the mining task. In general, given *F-list*$= x_1 \cdots x_n$, the set of max-patterns can be divided into n disjoint subsets: the i-th subset contains max-pattern having item x_i but none of x_j $(0 < j < i)$.

To mine max-patterns containing $X = x_{i_1} \cdots x_{i_m}$ (items in X are listed according to *F-list*), an *X-projected database* TDB_X is formed: every transaction $t = (tid, Y) \in TDB$ such that $X \subset Y$ is projected to TDB_X as (tid, Y'), only items after x_{i_m} in the *F-list* are in Y'. In Example 4, *F-list*$= a - b - c - d$. Thus, the *ac-projected database* TDB_{ac} contains only one transaction d (see Step 2). Here, the transaction-id is omitted.

The pruning techniques used in the mining are verified as follows.

First, *how can we determine whether a frequent pattern X is a local max-pattern?* We have the following lemma, while the proof is skipped due to lack of space.

Lemma 4.1 *Let X be a frequent pattern and $i_X = \max\{i \mid sup(X) \geq min\_sup_i\}$. Then, X is a max-pattern w.r.t. $min\_sup_{i_X}$ if and only if X is not a sub-pattern of any max-pattern w.r.t. $min\_sup_{i_X}$ and $sup_{TDB_X}(x) < min\_sup_{i_X}$ for each item x in TDB_X.*

In Step 1.b of Example 4, pattern ab is determined as a max-pattern w.r.t. support threshold 4 according to Lemma 4.1.

Second, *can we prune some unpromising patterns as early as possible?* We have the following lemma.

Lemma 4.2 *Let X be a frequent pattern and* F-list$_X = y_1 - \ldots - y_m$ *be the* F-list *of local frequent items in TDB_X. For an item y_i in* F-list$_X$*, if there exists a max-pattern Z and i $(1 \leq i \leq n\_level)$ such that $(X \cup y_i \cdots y_m) \subseteq Z$ and*

$$min\_sup_i \leq sup(Z) \leq sup_{TDB_X}(y_i) < min\_sup_{i+1}$$

then for $Y \subseteq y_i \cdots y_m$, $X \cup Y$ cannot be a max-pattern, and thus $(X \cup y_i)$-, ..., $(X \cup y_m)$-projected databases can be pruned.

Proof. We only need to notice the following two facts: (1) for X and y_i as stated in the lemma, $X \cup y_i \cdots y_m \subseteq Z$ is not a max-pattern, which follows Lemma 4.1, and (2) from X and $y_i \cdots y_m$, we cannot derive any max-pattern which is a super-pattern of Z, since $X \cup y_i \cdots y_m \subseteq Z$. Thus, we have the lemma. ∎

In Step 2 of Example 4, we do not need to form and mine bc-projected database since (1) the frequent items in b-projected database are c and d with support less than 4; and (2) bcd is not a max-pattern w.r.t. support threshold 1. Thus, Lemma 4.2 is applied here.

Based on the above analysis, we summarize the algorithm for constructing an M-base as follows.

Algorithm 2 *(*CFP-M*: a method for mining max-patterns at various layers)*

Input: transaction database TDB, support threshold min_sup, and error bound $k\%$;

Output: an M-base $\mathcal{B}$ w.r.t. ζ;

Method: Let I be the set of all items; call $mine(TDB, \emptyset, I)$.

Function $mine(DB_X, X, I_X)$

// DB_X: a projected database, X: a frequent pattern, I_X: a set of items to be processed

1. scan DB_X once to find all frequent items within I_X;
2. let F_e be the set of items appearing in every transaction in DB_X, i.e., $F_e = \{x \mid x \in I_X, sup(x) = |DB_X|\}$; let $F_r = F_X - F_e$;
3. let $i = \max\{j \mid |DB_X| \geq min\_sup_j\}$;
   if $min\_sup_i > sup(y)$ for each item $y \in F_r$, and $X \cup F_e$ is not contained in any max-pattern w.r.t. support threshold min_sup_i, then output $X \cup F_e$;
   // $X \cup F_e$ is a max-pattern w.r.t. min_sup_i. This step is based on Lemma 4.1.
4. let *F-list*$_X$ be the list of items in F_r in support descending order;
   for each item $x \in$*F-list*$_X$ (processed in the order) do
   (a) if the pruning criteria of Lemma 4.2 is satisfied for X, x (as y_i), and F_r (as F-list$_X$), then return;
   (b) otherwise, let $DB_{Xx} \subset DB$ be the subset of transactions containing x;
   let $I_{Xx} \subset F_r$ be the set of frequent items after x in F_X;
   call $mine(DB_{Xx}, F_e \cup \{x\}, I_{Xx})$;
5. return;

Analysis. The correctness of the algorithm follows the lemmas having shown before. In this algorithm, we do not check every frequent pattern. Instead, we only check frequent patterns without a proper super-pattern having exact same support count. Furthermore, by using Lemma 4.2, we prune patterns approximately contained by other max-patterns. ∎

The implementation of Algorithm 2 involves projected databases and containment tests of frequent patterns. Accordingly, we propose the following two implementation optimizations.

First, we use FP-tree [8] to compress database and projected databases. An FP-tree is a prefix tree storing transactions. Only frequent items in transactions are stored. From FP-trees, projected databases can be derived efficiently.

Second, one critical implementation issue of Algorithm 2 is that we need to identify max-patterns containing a given pattern and staying in the same support range. In our implementation, we index max-patterns of the condensed FP-base by support level i (i.e., the pattern is w.r.t. min_sup_i) and length. Moreover, to facilitate the search, we organize all max-patterns using a prefix tree, while all nodes with same item label are linked together.

5 Empirical Evaluation

To evaluate the effectiveness and efficiency of condensed FP-bases, we conducted a comprehensive set of experiments. In this section, we report a summary of our results. All experiments are conducted on a PC with Pentium III-750 CPU and 188Mb main memory. All the programs are coded using Microsoft Visual C++6.0. We use both synthetic datasets and real datasets in the experiments. The results are consistent. Due to lack of space, we only report results on three datasets as follows.

To report results on effectiveness and efficiency of FP-bases, we use two dense datasets, *Mushroom* and *Connect-4*, from the UC-Irvine Machine Learning Database Repository. A dataset is dense if it contains many long patterns even though the support threshold is relatively high. The *Mushroom* dataset contains 8124 transactions, while the average length of transaction is 23. The *Connect-4* dataset has 67557 transactions and each transaction has 43 items. Both of them are typical dense datasets. Mining frequent patterns from dense databases is very challenging.

To test the scalability of FP-bases, we also use a synthetic dataset $T10I4D100 - 1000k$. This dataset is generated using the well-known IBM synthetic data generator [2]. It is a sparse dataset simulating the market basket data. The number of transactions in this dataset is up to 1 million.

In our experiments, we compare the following three algorithms for mining condensed FP-bases.

CFP-D: the level-by-level method for constructing condensed FP-base $\mathcal{B}_d$, i.e., Algorithm 1.

CFP-CLOSET: we adapt the CLOSET algorithm [14] to *CFP-CLOSET* for mining condensed FP-base $\mathcal{B}_m$ as follows. CFP-CLOSET finds frequent closed patterns and checks whether a frequent closed pattern is in $\mathcal{B}_m$ according to Lemma 4.1. It outputs only frequent closed patterns.

CFP-M: it is Algorithm 2, which finds condensed FP-base $\mathcal{B}_m$ with all pruning and optimization.

Effect of Compression

The compression effects of condensed FP-bases can be measured by compression ratio defined in Equation 1. Please note that the smaller the compression ratio, the better the compression effect.

First, we fix the support threshold and test the compression ratio with respect to various error bounds. The results on datasets *Mushroom* and *Connect-4* are shown in Figure 2 and Figure 3, respectively.

Here, the error bound is set as a percentage of the total number of transactions in the dataset. If there are 1000 transactions in the dataset, then an error bound of 0.1% means that the absolute error bound is 1.

It is clearly shown that condensed FP-base B_m can achieve much better compression ratio than B_d. For example, in dataset *Mushroom*, when the support threshold is set to 14%, there are in total $103,845$ frequent patterns, and $2,591$ frequent closed patterns. As shown in Figure 2, B_m is much smaller than B_d. For both condensed FP-bases, the larger the error bound, the better the compression ratio.

Note when error bound is 0%, B_m is exactly the set of frequent closed patterns. As can be seen, frequent closed itemsets can achieve a good compression ratio. Condensed FP-base B_m can carry the benefit and take the advantage of error bound to do an even better compression.

Since condensed FP-base B_m performs better than B_d, we now focus on the compression effect of B_m with respect to support threshold. The results are shown in Figure 4 and 5, respectively.

To help verify the compression effect, we also plot the compression ratio of condensed FP-base using frequent closed patterns. A condensed FP-base using frequent closed patterns is with an error bound 0. As clearly shown in the two figures, the larger the error bound, the better the compression. The results also confirm that, even with some small error bound, condensed FP-base B_m can be much smaller than the condensed FP-base of frequent closed patterns.

The compression ratio also is sensitive to the distribution of frequent patterns with respect to a specific support threshold. Fortunately, the general trend is that the lower the support threshold, the better the compression. When the support threshold is low, there are many frequent patterns with similar support counts. Thus, one pattern in a condensed FP-base may be a "representative" of many patterns.

Similar trends can be observed for the compression effect of B_d, but the compression ratio of B_d is larger than that of B_m in the same setting, i.e., the compression power of B_d is weaker.

Efficiency of computing condensed FP-bases

We compare the runtime of *CFP-D*, *CFP-CLOSET* and *CFP-M* with respect to various error bounds in Figure 6. The support threshold is set to 93%. From the figure, we can see that the trends are as follows. The runtime of both *CFP-D* and *CFP-CLOSET* are insensitive to the error bound. The two methods find the complete set of frequent patterns and frequent closed patterns, respectively, which are their dominant costs. We note that the cost of computing $X.ub$ for pattern X in *CFP-D* and that of the super-pattern checking in *CFP-CLOSET* are very minor comparing to the expensive pattern mining in these two algorithms.

CFP-D fully utilize the error bound to prune the search space. The larger the bound, the faster the execution. Thus, it is faster than the other two algorithms when the error bound is not very small.

We observe a similar trend on dataset *Mushroom*. Limited by space, we omit the details here. Moreover, since *CFP-D* is dramatically slower than *CFP-CLOSET* and *CFP-M*, in the remainder of this subsection, our discussion focuses on *CFP-CLOSET* and *CFP-M*.

In Figure 7, we compare the runtime of *CFP-CLOSET* and *CFP-M* with respect to support threshold. The error bound is set to 0.1% of the total number of transactions in the dataset. When the support threshold is high, the runtime of both methods are close. However, when the support threshold is low, the runtime of *CFP-CLOSET* increases dramatically, since it has to mine and check the complete set of frequent closed patterns. The runtime of *CFP-M* increases moderately even when the support threshold is low, since the pruning techniques help confine the search in a small subset of frequent closed patterns.

Again, the similar trends are observed in experiments on other datasets. We omit the details here.

Scaling-up Test

We also test the scalability of condensed FP-bases as well as related algorithms.

First, we test the scalability of compression ratio of condensed FP-bases. (If the curve is flatter, we say that the curve is more scalable, since the compression ratio is not sensible to the database size.) In Figure 8, we show the results on dataset *Connect-4*. We fix the support threshold as 90% of the number of transactions in the tests, and vary the number of transactions from 10% to 100% of that in the original dataset. In the figure, we compare the compression ratio of an FP-base using frequent closed patterns and B_m. Interestingly, as the number of transactions increases, the compression ratio also increases. The reason is that, when there are more transactions, there are more patterns with various support. Thus, the compression effect is not as good as that in the databases with small numbers of transactions. Fortunately, both the number of frequent closed patterns and that of patterns in B_m do not increase dramatically. Moreover, B_m is more scalable, since its compression ratio increases in a more moderate way.

Second, we use the synthetic dataset $T10I4D100 - 1000k$ to show the scalability of Algorithm *CFP-M*. To make a comparison to the traditional frequent pattern mining, we include the runtime of CLOSET in the figure. CLOSET computes the set of frequent closed patterns. The results are shown in Figure 9. In this test, the error bound for *CFP-M* is set to 0.1%. From the figure, we can see that both methods are scalable with respect to the number of transactions in the datasets. Their runtime are also close. CLOSET is faster when the database is large, since it does not need to check against the error bound. *CFP-M* has a scalability comparable to CLOSET and, at the same time, achieves non-trivial compression.

In summary, from the experimental results, we can draw the following conclusions. First, condensed FP-bases can achieve non-trivial compression for frequent patterns. B_m often performs considerably better than B_d, and thus is more preferable. Second, the larger the error bound, the more we compress. Error bound can help to make the condensed FP-bases more compact. Third, *CFP-M* is an effi-

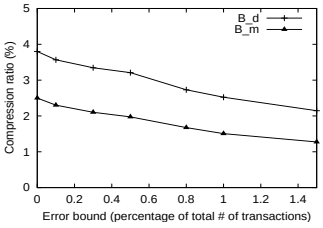


Figure 2. The compression ratio of B_d and B_m on dataset *Mushroom* ($min\_sup = 14\%$).

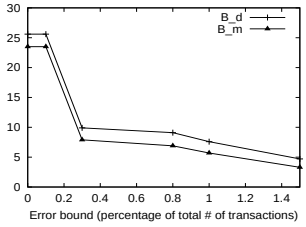


Figure 3. The compression ratio of B_d and B_m on *Connect-4* ($min\_sup = 93\%$).

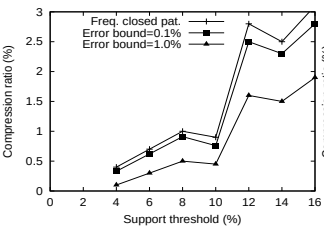


Figure 4. The compression ratio of B_m w.r.t. support threshold on dataset *Mushroom*.

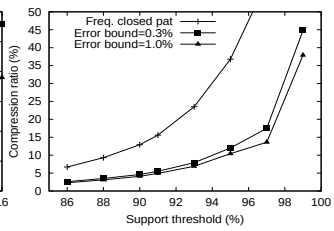


Figure 5. The compression ratio of B_m w.r.t. support threshold on dataset *Connect-4*.

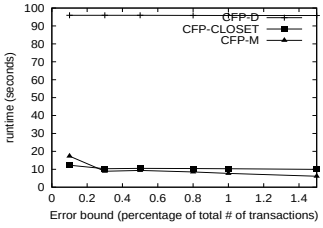


Figure 6. The runtime w.r.t. error bound on dataset *Connect-4* ($min\_sup = 93\%$).

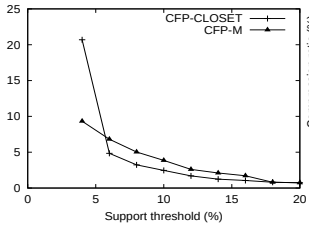


Figure 7. The runtime w.r.t. support threshold on dataset *Mushroom* ($err\_b = 0.1\%$).

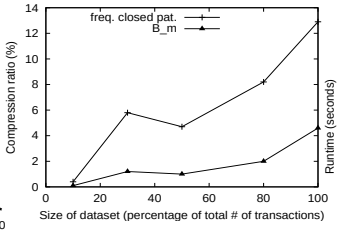


Figure 8. The scalability of compression ratio on *Connect-4* ($min\_sup = 90\%$).

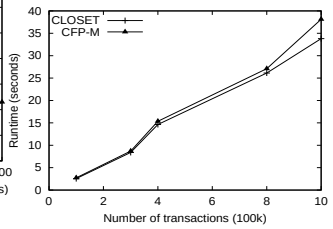


Figure 9. The scalability of runtime on dataset $T10I4D100 - 1000k$.

cient and scalable algorithm for computing condensed FP-base B_m. It is comparable to CLOSET in terms of runtime and scalability, and $\mathcal{B}_m$ achieves better compression effect than the set of all frequent closed patterns. The optimization and pruning techniques help make *CFP-M* efficient and scalable. Overall, B_m and *CFP-M* are the clear winners for frequent pattern base compression and corresponding computation.

6 Conclusions

In this paper, we introduced and considered the problem of mining a condensed frequent pattern base. The notion of condensed FP-base is introduced to significantly reduce the set of patterns that need to be mined, stored, and analyzed, while providing guaranteed error bound for frequencies of patterns not in the bases. We considered two types of condensed FP-bases: the downward condensed FP-base $\mathcal{B}_d$ and the max-pattern based condensed FP-base $\mathcal{B}_m$. Interesting algorithms and several novel optimization techniques are developed to mine condensed FP-bases. Experimental results show that we can achieve substantial compression ratio of condensation using the condensed FP-bases, and our algorithms are efficient and scalable. We also discussed some interesting extensions of our methods. As future work, it would be interesting to explore other effective condensed FP-bases and efficient mining methods.

Acknowledgements. The work was supported in part by U.S. NSF IIS-02-09199, Microsoft Research, University of Illinois, and NSERC and NCE/IRIS of Canada. The authors would like to thank the anonymous reviewers' comments which help improve the quality of the paper.

References

[1] R.C. Agarwal, C.C. Aggarwal, V. V. V. Prasad. Depth first generation of long patterns. In *KDD'00*.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB'94*.

[3] R. J. Bayardo. Efficiently mining long patterns from databases. In *SIGMOD'98*.

[4] J.-F. Boulicaut, A. Bykowski, C. Rigotti. Approximation of frequency queris by means of free-sets. In *PKDD'00*.

[5] D. Burdick, M. Calimlim, J. Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *ICDE'01*.

[6] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proc. KDD'99*.

[7] K. Gouda and M.J. Zaki. Efficiently mining maximal frequent itemsets. In *ICDM'01*.

[8] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD'00*.

[9] L. V. S. Lakshmanan, R. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *SIGMOD'99*.

[10] H. Mannila and H. Toivonen. Multiple uses of frequent sets and condensed representations. In *KDD'96*.

[11] R. Ng, L. V. S. Lakshmanan, J. Han, A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *SIGMOD'98*.

[12] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT'99*.

[13] J. Pei, J. Han, L. V. S. Lakshmanan. Mining frequent itemsets with convertible constraints. In *ICDE'01*.

[14] J. Pei, J. Han, R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *DMKD'00*.

[15] M. Zaki. Generating non-redundant association rules. In *KDD'00*.