

# Efficient Computation of Partial-Support for Mining Interesting Itemsets

Ardian Kristanto Poernomo\*

Vivekanand Gopalkrishnan†

## Abstract

Mining interesting itemsets is a popular topic in the data mining community. The objective of this problem is to mine all interesting itemsets, with respect to a given interestingness measure. While considerable efforts have been spent on justifying the various interestingness measures, the algorithms that mine them are not quite well-studied, except in the case *support*, which has resulted in the famous *frequent itemset mining* (FIM) problem.

In this paper, we show that a certain *class* of interesting itemsets can be represented by functions of their *partial support*. This class includes some definitions of fault-tolerant itemsets, *estimated support* of itemsets in noisy data, and *bond* of itemsets. As the name implies, partial support of an itemset is the number of transactions containing *some part* of the given itemset. This paper addresses the problem of efficiently calculating partial supports, which leads to efficient algorithms for mining interesting itemsets in that class. We show that there exists a recurrence relation between partial supports. Hence, we can calculate the partial supports of itemset by simply extending any FIM algorithm (even the implementation). This allows us to benefit from innovations and optimizations in FIM algorithms.

Theoretical analysis shows that our approaches retain the running time complexity of the base FIM algorithms for only a small cost in space. Extensive experiments on several real-world datasets also demonstrate that algorithms based on our approach are significantly faster than previously proposed techniques for corresponding definitions.

## 1 Introduction.

The objective of data mining is to discover knowledge from databases. Interesting itemset mining is not an exception, in which the knowledge is formed as sets of items (itemset) of a transactional database. The objective of this problem is to mine all interesting itemsets, with respect to an interestingness measure. The most famous variant of the interesting itemset mining problem is the well-known frequent itemset mining (FIM) problem [1]. In this problem, the interestingness of an itemset is measured as the number of transactions containing that entire itemset, which is also called as the *support* of that itemset. The higher the support, the more interesting the itemset, and in general, we're *interested* in itemsets whose support exceeds a certain threshold.

Nowadays, there are several popular interestingness measures for itemsets. Besides support, interestingness of itemsets have been defined based on the principle of information-theory [20, 12], statistical significance [10, 5, 15], and fault-tolerant paradigm [16, 21, 19]. These interestingness measures are well-established, and provably extract interesting knowledge from databases. In general, an interestingness measure of an itemset can be formulated as a function of its subsets' support. Some definitions [20, 5, 14, 9], require the support of all subsets to calculate the interestingness of an itemset  $X$ , yielding a lower bound of  $\Omega(2^{|X|})$  for any mining algorithm. In other definitions like that of FIM and *all-confidence* [15], the interestingness can be calculated trivially. In this paper we observe another class of definitions, where the interestingness measure can be formulated as a function of its *partial support*. This class of definitions include some of fault-tolerant itemsets (FTFP [16], strong/weak ETI [21], dense itemset [19]), estimated support of itemsets in noisy data [18], and *bond* of itemsets [15]. Therefore, if we can calculate the *partial supports* of itemsets efficiently, then we will be able to mine those interesting itemsets efficiently.

*Partial support* is our new term which represents the number of transactions containing *certain number of items* of an itemset. This *number* is specified by a parameter  $C$ . For example, partial support of an itemset  $X$ , where  $C = 2$ , represents the number of transactions that contain *exactly* 2 items of  $X$ . Similarly, partial support of  $X$  given  $C \leq 3$  represents the number of transactions containing *at most* 3 items of  $X$ . *Partial support* generalizes the standard definition of *support*, since the *support* of an itemset  $X$  is equivalent to its *partial support* given  $C = |X|$  or  $C \geq |X|$ . The formal definition of partial support is given in Section 3.1.

While no previous publications have mentioned partial support, we note that algorithms for mining fault-tolerant frequent pattern (FTFP) [16] can be borrowed to calculate partial supports of itemsets. However, analyzing previously proposed algorithms for mining FTFP, we find that they are significantly less efficient compared to the usual FIM algorithms. This inefficiency is

\*Nanyang Technological University, Singapore.

†Nanyang Technological University, Singapore.

discussed in detail in Section 4.4. This efficiency issue leads us to research on efficient algorithms for calculating partial supports.

We discover that a recurrence relation exists between partial supports of an itemset  $X$  and its *direct subsets*, i.e., subsets of size  $|X| - 1$ . For example, partial support of an itemset  $abc$  can be calculated by knowing only the support of  $abc$ , and the partial support of itemsets  $ab$ ,  $ac$ , and  $bc$ . Based on this recurrence relation, we can reuse any FIM algorithm – even the implementation, to mine interesting itemsets. The time complexity of this approach is comparable to that of the base FIM algorithm. The trade-off is in space requirement, since we need to record all frequent itemsets, as well as their partial supports, which is  $O(|X|)$  per interesting itemset  $X$ . However, with current technology, the memory requirement is no longer an issue. Furthermore, we can ignore cases where the results contain billions of itemsets, since they are simply infeasible to analyze.

Summarizing, we address two problems in this paper. The first problem is to show that partial supports can indeed derive many definitions of interesting itemsets, and the second is to design an efficient algorithm for calculating partial supports. To those ends, we:

- introduce partial supports of itemsets, and show how they can be used to derive some definitions of interesting itemsets.
- prove a recurrence relation between partial supports of an itemset and its direct subsets.
- design very efficient algorithms, and more importantly very similar to usual FIM algorithms – even in implementation level, to calculate partial supports of itemsets.

Our claims on the efficiency of our algorithms are validated by extensive experiments. First, we evaluate our extensions of standard FIM algorithms for the task of calculating partial supports. In general, the time taken by our approaches is comparable to that of the original FIM algorithms they are based upon, while other competing algorithms are significantly less efficient. Second, we use our algorithms to mine various definitions of interesting itemsets, viz., Dense Itemset, FTFFP, and MASK. Noting that the original algorithms are very similar to the naïve algorithms for calculating partial supports, it is not surprising that our algorithms are significantly faster compared to them. Regarding space requirement, our algorithms need additional space for storing all interesting itemsets in which other algorithms might not need. Nevertheless, as we mentioned previously, space requirement is no longer an issue in current computing technologies.

The rest of this paper is organized as follows. Section 2 briefly describes several definitions of interesting itemsets and relates one such class with partial supports. Section 3 presents basic notations, and discusses the process of deriving interestingness measures using partial supports. Previous algorithms to calculate partial supports are presented in Section 4. Section 5 presents the recurrence relation which is the basis of our approach, followed by the algorithms in Section 6. Our empirical evaluations are presented in Section 7. Finally, section 8 concludes with future works.

## 2 Related work.

### 2.1 Interesting Itemsets.

Measuring interestingness of itemsets has been an active topic in data mining community. The pioneering technique [1] relates the interestingness of itemsets to statistical analysis, i.e., an itemset is considered interesting if the probability of its occurrence is high. Thereafter, several other definitions for interesting itemsets have been proposed, based on principle of information-theory [20, 12], statistical significance [5, 10, 15], and fault-tolerant paradigm [16, 21, 19]. These interestingness measures are well-established, and proven able to extract interesting knowledge from databases. However, while every paper has justified its own interestingness measure, not much effort is spent on improving the mining approach, which is the focus of this paper.

Despite these variations, all interestingness measures of an itemset can be formulated as functions of its subsets' support. Some definitions need the support of all subsets, like [5] that uses  $\chi^2$  independence test, [20] that compares the support with maximum-entropy distribution, and some variants of fault-tolerant itemsets, viz., AFI [14] and AC-Close [9]. Since these definitions are based on all the subsets' supports, the lower bound complexity for calculating the interestingness of an itemset  $X$  is  $\Omega(2^{|X|})$ . In other words, it is impossible to design a very efficient algorithm for calculating the interestingness of itemsets. Hence, this class of interestingness measure is not the focus of this paper.

In some other definitions, not all subsets' supports are necessary. For example in FIM, only the support of the itemset is needed. In all-confidence [15], only the support of the itemset, and all 1-subsets (subset with size 1) are needed. With such a lenient requirement, the interestingness of an itemset can be calculated trivially. Hence, this class of interestingness measure is not the focus of this paper.

We note that there is a class of interestingness measures, where only the number of transactions containing a *certain number of items* are important. In such cases, if the itemset is  $abc$ , the interestingness can be

formulated as a function of the number of transactions containing exactly 0, 1, 2, and 3 items of  $abc$ . With such a requirement, the time complexity for measuring the interestingness is lower bounded by  $\Omega(|X|)$ , which is quite a small number. This class of problems constitute the focus of this paper.

Examples of measures belonging to this class are fault-tolerant itemsets (FTFP [16], strong/weak ETI [21], dense itemset [19]), estimated support of itemsets in noisy data [18], and *bond* of itemsets [15], which is the ratio between the number of transactions that contain *all* items, and those which contain *any* item of the itemset.

These measures have been proven to be useful in a broad range of data mining applications. Yang et al. [21] showed that their measures can discover itemsets that are better than usual frequent itemsets for EM clustering. Pei et al. [16] showed that they can mine *generalized association rules* using their notion of interesting itemsets. Rizvi and Haritsa. [18] showed that their measure enables accurate privacy-preserving itemset mining. Bond of itemset is analogous with *Jaccard similarity* in information retrieval systems. Omiecinski [15] shows that *bond* can indeed be used to extract interesting knowledge from real-world databases, which cannot be discovered using usual support. The paper also demonstrates that their algorithm to calculate *bond* is much slower than that to calculate *all-confidence*. Besides these measures, we also sketch several new variants of interestingness measures belonging to this class.

Recently, [11] has compared some interestingness measures for fault-tolerant itemsets. However, as stated earlier, our objective is to show that partial supports can be used to efficiently derive some interestingness measures, and not to propose or compare other measures.

## 2.2 Partial Support.

To the best of our knowledge, there is no previous work mentioning partial supports of itemsets. However, we note that we can borrow algorithms for mining fault-tolerant frequent pattern (FTFP) [16], to calculate partial supports of itemsets. In FTFP, for a given itemset  $X$ , we calculate the number of transactions lacking at most  $\varepsilon$  items of  $X$ . This is equivalent with our notion of partial supports, where  $C \geq |X| - \varepsilon$ .

Several algorithms have been proposed to mine FTFP, viz., FT-Apriori [16] which extends Apriori algorithm for FIM, BIAS-T [17] which uses depth-first enumeration while calculating aggregation statistics, and VB-FT-Mine [13] that discovers the recurrence relation of transactions supporting each itemset. These algorithms are described in Section 4. However, as discussed in Section 4.4, they are significantly less

| TID | Itemset |
|-----|---------|
| 1   | a,b,c,d |
| 2   | a,b,e   |
| 3   | a,b     |
| 4   | c,d,e   |
| 5   | a,c,e   |
| 6   | b,c     |
| 7   | b,c,d   |
| 8   | c       |

Table 1: Sample transactional database

efficient compared to the usual FIM algorithms.

Partial supports may also be computed using the inclusion-exclusion principle, as is done to find the supports of all *generalized itemsets* [8]. Generalized itemsets are itemsets that may contain negated items, which *must not* appear in the transaction to be considered supporting them. For example, the support of itemset  $\bar{a}bc$  is calculated as the number of transactions containing items  $b$  and  $c$ , and not containing item  $a$ . The number of generalized itemsets of an itemset  $X$  is  $2^{|X|}$ , hence this technique is overkill to calculate partial supports. For comparison, the algorithm proposed in [8] has time complexity  $O(|X| \cdot 2^{|X|})$ , while ours is  $O(|X|^2)$ . Note that these complexities assume that the (partial) supports of all  $X$ 's subsets can be accessed in constant time, hence are slightly different from our analysis in Section 5.

## 3 Usability of partial support.

### 3.1 Basic notations.

In this paper, we deal with a transactional database  $D$ , which consists of transactions  $T$  over items  $I$ . Database  $D$  can also be treated as a binary matrix  $D = T \times I$ , where a cell  $D(y, x)$  has value 1 if transaction  $y$  contains item  $x$ , and 0 otherwise. The density of a matrix is calculated as the proportion of cells with value 1 within it. Function  $I(y)$  represents the itemset which is contained in transaction  $y$ . Similarly function  $T(x)$  represents the set of transactions (tidset) containing item  $x$ . An itemset of size  $k$  is also called  $k$ -itemset, similarly for tidset. A transaction  $t$  supports an itemset  $X$ , if it contains all items in  $X$ . For brevity, we sometimes write an itemset  $\{a, b, c\}$  as a sequence of characters  $abc$ .

The (*absolute*) support of an itemset  $X$ , denoted as  $sup(X)$ , is the number of transactions containing  $X$  in  $D$ . An itemset  $X$  is considered frequent if its support is no less than a user-specified threshold  $min\_sup$ . The ratio of *absolute support* to the total

number of transactions in the database is called *relative support*, which is also sometimes used. Unless explicitly stated, in this paper the term *support* relates to *absolute support*.

EXAMPLE 3.1. Table 1 presents a database with five items and eight transactions. The support of itemset  $ab$  is 3, since there are three transactions, viz., transactions 1, 2, and 3, that contain itemset  $ab$ ; while the support of itemset  $bcd$  is 2 since there are two transactions, 1 and 7, that contain it.

Partial support of an itemset  $X$  is the number of transactions satisfying *criteria*  $C \in \{\leq k, = k, \geq k\}$ , and is denoted as  $sup_C(X)$ , where  $k$  is an integer between 0 and  $|X|$ . Criterion ( $C \leq k$ ) is satisfied by transactions containing *at most*  $k$  items of  $X$ . Similarly, criteria ( $C = k$ ) and ( $C \geq k$ ) are satisfied by transactions containing *exactly*  $k$  and *at least*  $k$  items of  $X$  respectively.

EXAMPLE 3.2. Considering the database in Table 1, we observe that  $sup_{=2}(abc) = 5$ , corresponds to tidset  $\{2,3,5,6,7\}$ , each of which contain *exactly* 2 items of  $abc$ . Similarly,  $sup_{\geq 2}(abc) = 6$  corresponds to tidset  $\{1,2,3,5,6,7\}$ , each of which contain *at least* 2 items of  $abc$ .

Note that calculating one particular *type* of partial support, for example ( $C \geq k$ ) for all  $k$ , is sufficient, since the remaining can be derived easily as:

$$sup_{=k}(X) = sup_{\geq k}(X) - sup_{\geq k+1}(X)$$

and

$$sup_{\leq k}(X) = |T| - sup_{\geq k+1}(X)$$

### 3.2 Previous definitions of interesting itemsets.

FTFP [16] and Strong-ETI [21] are definitions of fault-tolerant itemsets that allow each transaction to lack several items. The difference is that FTFP allows *fixed* number of errors, while Strong-ETI tolerates errors proportional to the size of itemset. However, this difference only affects the itemset enumeration algorithm, and not the interestingness calculation algorithm. Given itemset  $X$ , if the maximum error tolerated is  $k$ , the interestingness of an itemset can be calculated using partial supports as  $sup_{\geq |X|-k}(X)$ .

Weak ETI [21] and Dense Itemset [19] defines the interestingness of an itemset as the size of largest tidset  $Y$ , such that the density of matrix  $Y \times X$  is at least  $\delta$ , where  $\delta$  is a user-defined parameter. The difference between these two is that the latter imposes an additional constraint requiring all subsets of a *dense*

*itemset* to also be dense itemsets. The algorithm [19] to find this tidset first calculates the *intersection profile* of itemsets. Having the intersection profile, the time complexity to calculate the interestingness of itemset  $X$  is  $O(|X|)$ . Intersection profile is equivalent to  $sup_{=k}(X)$ , hence it can also be calculated using partial supports.

MASK [18] is a framework to estimate the support of itemset before the database was affected by *Bernoulli noise*. Bernoulli noise affects each cell in the whole matrix independently and identically. Bernoulli noise is usually defined with two parameters,  $p$  which represents the probability that a ‘1’ is correctly represented as ‘1’, and  $q$  which represents the probability that a ‘0’ is correctly represented as ‘0’. An itemset’s support estimation can be calculated using its partial support, as:

$$(3.1) \sum_{k=0}^{|X|} sup_{=k}(X) \cdot \left(\frac{q}{p+q-1}\right)^k \cdot \left(\frac{q-1}{p+q-1}\right)^{|X|-k}$$

*Proof.* See Appendix A.

Another interestingness measure is called *bond* [15] which is defined as the ratio between the number of transactions containing *all items*, and the those that contain *any item*. The former is  $sup_{=|X|}(X)$ , and the latter is  $sup_{\geq 1}(X)$ . We can also extend this definition into *generalized bond*, by replacing the latter with  $sup_{\geq k}(X)$ , or  $sup_{\geq |X|-k}(X)$ .

### 3.3 Other potential definitions of interesting itemsets.

Having partial supports of an itemset, many intuitive definitions of interesting itemsets can be derived. For example, the density of the matrix  $T \times X$  can be calculated as

$$(3.2) \frac{1}{|T| \cdot |X|} \cdot \sum_{i=0}^{|X|} sup_{=i}(X) \cdot i.$$

It is also possible that analysts wish to give *weightage* for transactions containing a certain number of items within an itemset. Hence, Equation 3.2 can be generalized as

$$(3.3) \sum_{i=0}^{|X|} sup_{=i}(X) \cdot w(i, |X|).$$

Note that the interestingness formula of strong-ETI and MASK is also a specialized form of Equation 3.3. While our objective is not to define a proper interestingness measure, we show that partial supports

of itemset can indeed be used to derive many variations of interestingness measures. Nonetheless, even without any interestingness measure, having partial supports of itemsets definitely provides more knowledge to the analyst compared to only the support.

#### 4 Previously proposed algorithms.

As mentioned in the Introduction, even though no prior works deal with partial support, we note that partial supports of itemsets can be calculated using algorithms that mine FTFP. Here, we present several such algorithms, which have been slightly modified for this task.

##### 4.1 Naïve Algorithm.

This algorithm maintains  $|X| + 1$  counters, representing the number of transactions containing 0 to  $|X|$  items. Given itemset  $X$ , this algorithm iterates each transaction, calculates the number of items contained by that transaction with respect to  $X$ , and increases the counter of transactions containing that many items. The time complexity of this algorithm is  $O(|T| \cdot |X|)$ .

##### 4.2 Counting Algorithm [17].

This algorithm is designed to be integrated with a depth-first enumeration of interesting itemsets [22]. Hence, we can assume that only one item is added or removed at one time.

In addition to  $(|X| + 1)$  counters as in naïve algorithm, this algorithm also maintains one counter per transaction, representing the number of items which are contained by the transaction, with respect to itemset  $X$ . When an item is added or removed, we iterate all transactions containing that item, and update the counters accordingly. The complexity of this algorithm is  $O(|T(a)|)$ , where  $a$  is the added or removed item.

##### 4.3 VB-FT-Mine [13].

Similar to the counting algorithm, this algorithm is also meant to be integrated with depth-first enumeration. Let  $S_{\geq k}(X)$  be the set of transactions containing at least  $k$  items of  $X$ , and  $a$  is any item in  $X$ . We have,

$$(4.4) \quad S_{\geq k}(X) = S_{\geq k}(X \setminus \{a\}) \cup (S_{\geq k-1}(X \setminus \{a\}) \cap T(a))$$

Now,  $sup_{\geq k}(X)$  is simply the cardinality of the set  $S_{\geq k}(X)$ . The time complexity to calculate  $sup_{\geq k}(X)$  is  $O(|T|)$  for one particular  $k$ , hence VB-FT-Mine runs in  $O(|X| \cdot |T|)$ . Note that since the only operations are set intersection and union, we can use a bit-set data structure which speeds up the operations by roughly 32 times in a 32-bits processor.

##### 4.4 Comments on inefficiency.

Depending on how they traverse frequent itemsets, FIM algorithms can be classified as Apriori algorithms which traverse level-by-level, and depth-first algorithms, which keep adding items, and backtrack once the itemset is no longer frequent.

**4.4.1 Apriori.** Advantages of Apriori algorithms are that the number of candidates (itemsets whose support is calculated) are relatively small, and the support of all candidates in the same level can be calculated at once. The state-of-the-art support counting procedure in Apriori algorithms is *recursive counting* [3], and roughly works as follows. First, all candidates are stored in a prefix-tree data structure, with each leaf corresponding to one candidate. For each transaction, we traverse the prefix-tree, then increment the counter whenever we reach the leaf of the tree.

This algorithm is efficient for FIM since we can prune the traversal whenever we surely cannot reach any leaf. However, for calculating partial supports, this pruning is not applicable, since we also calculate the number of transactions containing 0 item of  $X$ . For example, let the transaction contain itemset  $xy$ , and the candidate itemset be  $abcd$ . Even if  $xy$  does not intersect  $abcd$ , we still need to traverse  $abcd$  to increase the counter for number of transactions containing 0 item. Without such pruning, *recursive counting* has no benefit compared to the slow *naïve algorithm* (Section 4.1).

**4.4.2 Depth First Traversal.** In depth-first traversal, only one item is inserted or removed at a time. Hence, algorithms based on depth-first traversal are generally more efficient, since we can use the previous information. For example, we already have the information of an itemset  $X$  while calculating the support (or other information) for itemset  $X \cup \{a\}$ . Several optimization techniques have been developed for support-counting in depth-first traversal, like *database projection* [22] and *diffset* [23]. However, they are not applicable for calculating partial supports due to similar reasons as in Apriori. Moreover, in order to use bit-operations (section 4.3), we need to repeat the procedure  $|X|$  times to find the tidsets for each  $k$ , which *multiplies* the time complexity by  $O(|X|)$ . Algorithms without bit-operations (section 4.2) are also not efficient (for dense databases), especially because we cannot apply optimizations like database projection and diffset.

## 5 Recurrence relation of partial supports.

### 5.1 Derivation idea.

The idea of recurrence relation is based on *inclusion-exclusion principle*. If we know the support

of an itemset  $X$  and all its subsets, we can calculate the support of all *generalized itemsets* of  $X$  [6] (Section 2.1).

For example, the number of transactions containing item  $a$ , but not  $b$  and  $c$ ,  $sup(ab\bar{c})$ , can be calculated as  $sup(a) - sup(ab) - sup(ac) + sup(abc)$ . In general, the support of itemset  $X\bar{Y}$  can be calculated as:

$$(5.5) \quad sup(X\bar{Y}) = \sum_{Z \subseteq Y} (-1)^{|Z|} sup(XZ).$$

Knowing *generalized supports* of an itemset  $X$ , its partial support,  $sup_{=k}(X)$ , can be calculated as:

$$(5.6) \quad sup_{=k}(X) = \sum_{Y \subseteq X, |Y|=k} sup(Y\overline{(X \setminus Y)}).$$

By using only these equations, we can calculate the partial support of an itemset as a function of its subsets' support. Let  $Y \subseteq X$ , and  $|Y| \geq k$ . Combining Equations 5.5 and 5.6, we can calculate the coefficient of  $sup(Y)$  in  $sup_{=k}(X)$  as  $(-1)^{|Y|-k} \cdot \binom{|Y|}{k}$ . Let  $sum_{=k}(X) = \sum_{Y \subseteq X, |Y|=k} sup(Y)$ , hence equation 5.6 can be further simplified into:

$$(5.7) \quad sup_{=k}(X) = \sum_{i=k}^{|X|} (-1)^{i-k} \cdot \binom{i}{k} \cdot sum_{=i}(X).$$

Since  $sup_{\geq k}(X) = \sum_{i=k}^{|X|} sup_{=i}(X)$ , the coefficient of  $sup(Y)$  in  $sup_{\geq k}(X)$  can be calculated as  $\sum_{i=k}^{|Y|} (-1)^{|Y|-i} \binom{|Y|}{i} = (-1)^{|Y|-k} \binom{|Y|-1}{k-1}$ , by the identity of binomial coefficient. Hence, a similar function can also be derived for  $sup_{\geq k}(X)$ , which is:

$$(5.8) \quad sup_{\geq k}(X) = \sum_{i=k}^{|X|} (-1)^{i-k} \cdot \binom{i-1}{k-1} \cdot sum_{=i}(X).$$

Note that the coefficient of  $sup(Y)$  in  $sup_{=k}(X)$  is independent of  $X$ . For example, the coefficient of  $sup(ab)$  in  $sup_{=1}(abc)$  is equal to the coefficient of  $sup(ab)$  in  $sup_{=1}(abcd)$ . Hence, we suspect that there might be a simple relation between them.

## 5.2 Final formula.

Having analyzed the formula, we discover that a simple recurrence relation between partial supports indeed exists, which is presented in following equation.

$$(5.9) \quad sup_{\geq k}(X) = \begin{cases} sup(X) & \text{if } k = |X| \\ \frac{(\sum_{a \in X} sup_{\geq k}(X \setminus \{a\})) - k \cdot sup_{\geq k+1}(X)}{|X| - k} & \text{otherwise} \end{cases}$$

*Proof.* See Appendix B.

---

### Algorithm 1: Modified candidate generation for calculating partial support

---

```

1 let  $X$  be the candidate;
2 for  $a \in X$  do
3   for  $k = 0$  to  $|X| - 1$  do
4      $sup_{\geq k}(X)_+ = \frac{1}{|X| - k} \cdot sup_{\geq k}(X \setminus \{a\})$ ;
5   end
6 end

```

---

There is also a very similar recurrence relation for  $sup_{=k}(X)$ , which is:

$$(5.10) \quad sup_{=k}(X) = \begin{cases} sup(X) & \text{if } k = |X| \\ \frac{(\sum_{a \in X} sup_{=k}(X \setminus \{a\})) - (k+1) \cdot sup_{=k+1}(X)}{|X| - k} & \text{otherwise} \end{cases}$$

The difference is only in the coefficient  $k + 1$  in Equation 5.10 instead of  $k$  in Equation 5.9. We omit the proof as it is very similar to that for Equation 5.9.

Having this recurrence relation, the partial support of an itemset  $X$  can be calculated by knowing the partial support of each  $(|X| - 1)$ -subset of  $X$ , and the actual support of  $X$ . Note that the nature of recurrence relation presented here is different from that of VB-FT-Mine (Section 4.3). In VB-FT-Mine, the domain is the *supporting-tidsets*, while ours is the *partial supports* themselves. This difference affects the efficiency significantly. While in VB-FT-Mine we need to reiterate each transaction, which is  $O(|T|)$ , in our case we only need to do arithmetic manipulation, which is  $O(|X|)$ .

## 6 Interesting itemset mining algorithms.

Having the recurrence relation, the next step is to design the algorithm to calculate the partial supports. Instead of designing entirely new algorithms, we *modify* existing FIM algorithms to use that recurrence relation. This approach allows us to retain the simplicity and elegance of the algorithms, while the complication of computing partial support is handled by the formulation of the recurrence relation in the previous section. This section presents the modification of Apriori and Depth-first traversal to calculate partial supports of itemsets.

### 6.1 Apriori.

Since the recurrence relation is on the subset of each itemsets, implementing this idea for Apriori algorithms is straight-forward. During *candidate generation* procedure, we update the LHS of the partial supports in Equation 5.9 (the one inside summation of subsets' partial support). Once we have  $sup(X)$  after sup-

---

**Algorithm 2:** Modified support counting for calculating partial support

---

```

1  $sup_{\geq |X|}(X) = sup(X);$ 
2 for  $k = |X| - 1$  to 0 do
3    $sup_{\geq k}(X) = \frac{k}{|X|-k} \cdot sup_{\geq k+1}(X);$ 
4 end

```

---

port counting procedure, we can update  $sup_{\geq k}(X)$  iteratively. These procedures are presented in Algorithm 1 and Algorithm 2. Compared to Apriori algorithm for FIM, this algorithm increases the time complexity of several procedures by  $O(|X|)$ . However, assuming that the complexity is dominated by the support-counting procedure (which is usually valid since support-counting is in the order of  $|T|$ , while others are in  $|X|$ , and  $|X| \ll |I| \ll |T|$  in most cases), such increments are negligible.

*Proof.* Let the number of iterations for candidate generation be  $C(X)$  and that for support counting be  $S(X)$ . The total time complexity for Apriori algorithm for FIM, with respect to itemset  $X$ , is given as  $O(C(X) + S(X))$ . For calculating partial supports, the total time complexity becomes  $O(C(X) \cdot |X| + S(X) + |X|)$ . Assuming that  $C(X) \cdot |X| = O(S(X))$ , the overall time complexity doesn't change. ■

Analyzing the space complexity, this algorithm requires us to store the partial supports of all interesting itemsets for the last 2 levels, hence increases the space requirement by factor of  $O(|X|)$ . Note that  $|X|$  is generally a very small number.

## 6.2 Depth First Traversal.

Having the requirement to know the partial support of all subsets, it seems impossible to apply this idea with depth-first algorithms. For example, in depth-first traversal, we iterate itemset  $a$ , then  $ab$ , then  $abc$ . To use the recurrence relation, we need the results of both itemset  $a$  and  $b$  to calculate the partial supports of  $ab$ , which is not yet available since itemset  $b$  hasn't been traversed.

However, thanks to the result of [7], this technique can be implemented by reversing the order of depth-first enumeration and storing the partial supports of all frequent itemsets. The idea is to ensure that all subsets of an itemset have been traversed before that itemset is traversed. For more details, please refer to [7]. Having that condition, to apply the recurrence relation is straight-forward. For mining itemset  $X$ , prior to calculating the support, we check whether all direct subsets of  $X$  are frequent, while concurrently updating

the partial support (LHS in Equation 5.9, analogous with Algorithm 1). After calculating the support, we update the partial supports, as in Algorithm 2.

Complexity-wise, assuming the dominating procedure is that which calculates support, this algorithm doesn't increase the complexity compared to usual FIM algorithm.

*Proof.* Let the number of iterations for calculating the support be  $S(X)$ , and that to locate (to put) the itemset  $X$  inside (into) hash table be  $h(X)$ . The time complexity of usual FIM algorithm is  $O(S(X))$ , and after this extension becomes  $O(|X| \cdot h(X) + |X|^2 + S(X) + |X| + h(X))$ , for locating all subsets, initializing the partial supports, calculating support, updating the partial supports, and putting the itemsets into hashtable. ■

In our implementation, we use a simple hash function with *linear chaining* for collision detection. The space requirement of our algorithm is  $O(|X|)$  per interesting itemsets, for storing the itemsets themselves and the partial supports.

## 7 Experimental Result.

We performed several empirical evaluations. First, we calculate the partial supports of all frequent itemsets. We compare the efficiency of our algorithms with other methods to calculate partial support, as well as the original algorithms for FIM. Next, we use our algorithms to mine FTFP [16], Dense Itemset [19], and MASK [18], and compare with the previously proposed algorithms to mine those itemsets.

All experiments are run in Pentium 4, 3 GHz machine, with 1 GB main memory, and all codes are implemented in C++. We implemented the base algorithm, Apriori and Eclat, ourselves to avoid bias in implementations. Our algorithms can be easily integrated with *any* FIM implementation. However, some other algorithms cannot since they are not applicable with some optimization techniques.

We use four datasets, namely BMS-Webview-1 [24], Retail [4], Mushroom, and Chess, representing sparse and dense datasets. All can be freely downloadable in FIMI website<sup>1</sup>. Some characteristics of datasets are presented in Table 2.

The experiment is conducted as follows. Each algorithm is run on each dataset with varying minimum supports. The time taken for each run is noted, and executions that exceed one hour are terminated. Note that minimum supports provided in the charts

<sup>1</sup><http://fimi.cs.helsinki.fi/data/>

| Name          | $ T $  | $ I $  | Max. length | Avg. length | Density |
|---------------|--------|--------|-------------|-------------|---------|
| BMS-Webview-1 | 59,602 | 497    | 267         | 2.511       | 0.51%   |
| Retail        | 88,162 | 16,470 | 76          | 10.306      | 0.06%   |
| Mushroom      | 8,124  | 119    | 23          | 23          | 19.33%  |
| Chess         | 3,196  | 75     | 37          | 37          | 49.33%  |

Table 2: Datasets for experiment

are stated as *relative support*, and the time taken is presented in logarithmic scale.

### 7.1 FIM.

In this experiment, the targets of mining are frequent itemsets. The original Apriori and Eclat implementations are stated as APRIORI and ECLAT respectively. Those implementations are optimized. Apriori implementation uses *recursive counting* procedure, and Eclat implementation uses *Database Projection*, *Diffset*, *Dynamic reordering*, and *sparse bitset* data structure. Details of these techniques can be referred to [3]. Our extensions to calculate partial supports are denoted as APRIORI-PS and ECLAT-PS respectively. Previous techniques are denoted as AN-PS (Naïve algorithm, Section 4.1), BIAS-PS (counting algorithm, Section 4.2), and VB-FT-MINE (Section 4.3). The results are depicted in Figure 1.

It is clear from Figure 1 that our algorithms to calculate partial support are always the fastest in all datasets. It is observed that un-optimized Apriori (AN-PS) is faster than our optimized one in *Chess* dataset. The reason is because the optimization of Apriori algorithm is not meant to be applied in dense datasets, where ECLAT algorithm is more suitable. In such cases, it even slows down the algorithm.

Another interesting observation from the plots is that our methods are comparable with normal FIM algorithms. In *BMS-Webview-1*, ECLAT-PS is even slightly faster than ECLAT. Prior to calculate the support, ECLAT-PS verifies whether all direct subsets are frequent. Therefore, one explanation of this phenomena is that ECLAT did much more support-counting procedure than ECLAT-PS. On the contrary, in *chess*, ECLAT-PS takes roughly twice as much time as ECLAT. A possible reason is that when the results are plenty, the time to access the hash-table becomes non-negligible.

Regarding space requirement, our algorithms need  $O(|X|)$  space for storing all itemsets, as well as the partial supports. In our worst test case (*chess* with minimum support 40%), the number of results is around 7 million, and the total space required for the hash table is around 450 MB, which stills fits into main memory.

In most other cases, the requirements are still in the order of tens of megabytes.

### 7.2 Dense Itemset.

The targets of mining in this experiment are *Dense Itemsets* [19]. In order to avoid bias in implementations, we implemented the algorithm (which is faster than the original implementation). The results for this experiment is depicted in Figure 2. In Figure (a) to (d), we vary minimum support, while fixing  $\delta$  to 1. In Figure (e) to (h), we vary  $\delta$  while fixing minimum support. The parameter  $\delta$  is chosen in such a way such that the difference is noticeable. Datasets like BMS-Webview-1 and Chess are very sensitive to  $\delta$ , and hence we can only lower down to 90%. Other datasets are not that sensitive, hence allow smaller  $\delta$ .

Our algorithms are mentioned as ECLAT-PS and APRIORI-PS, and the original algorithm is mentioned as DENSE-SETS. DENSE-SETS is very similar to Apriori-Naïve algorithm, hence the result is also similar to that of the previous experiment (Section 7.1).

### 7.3 MASK.

Here, the interestingness is measured as the estimated true support, given a noisy database [18] (Section 2.1). We use the optimized algorithm described in [2], which is denoted as EMASK. The EMASK algorithm is similar to that which mines non-derivable itemsets [6]. Since it needs at least  $2^{|X|}$  iterations for calculating the interestingness of itemset  $X$ , this algorithm is expected to be inefficient, especially for large itemsets. Here, we integrate their technique with our optimized Apriori implementation.

The results for this experiment are depicted in Figure 3. In Figure (a) to (d), we vary minimum support while fixing  $p$  and  $q$  to 1 (no noise). In Figure (e) to (h), we vary the noise parameters  $p$  and  $q$ , while fixing minimum support. Note that we do not actually contaminate the dataset with noise, but simply assume that they are noisy. Hence, the result is deterministic. In any case, the assumption does not matter since we only measure the efficiency, and not the accuracy.

For simplicity, we set  $p$  equals to  $q$ . Hence, we only have one parameter to vary. As in previous experiment,

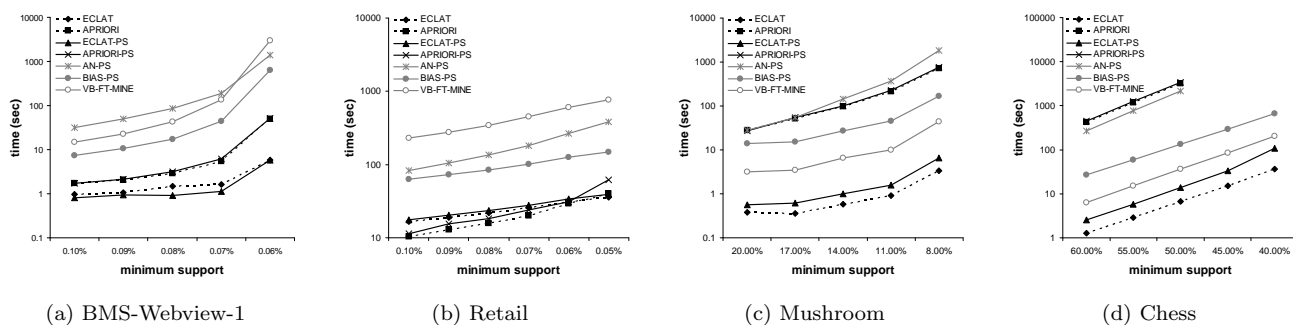


Figure 1: Running time for mining frequent itemsets

the value of  $p$  and  $q$  are set in a way such that the difference is noticeable. The impact of  $p$  and  $q$  varies between datasets. For example, *BMS-Webview-1* and *Retail* are very sensitive to  $p$  and  $q$ . When these parameters are set to be as low as 99%, there are no frequent itemsets remaining. *Mushroom* dataset is not sensitive to these parameters. In *Chess* dataset, the effect is even reversed, i.e., lower  $p$  and  $q$  results in more itemsets.

The outcomes are as expected, that EMASK is always slower than APRIORI-PS, especially for lower minimum supports (those with more and longer results).

#### 7.4 FTFP.

Mining FTFP is slightly more complicated compared to other variants, since they require all *items* to be *frequent* inside the *supporting transactions*, i.e., all items must appear in at least  $min\_sup^{item}$  transactions. However, the frequency of an item within supporting transactions can also be calculated using recurrence relation of partial supports. If  $sup_{=k}(a|X)$  is the number of transaction containing exactly  $k$  items of  $X$ , and one of them is item  $a$ , then

$$(7.11) \quad sup_{=k}(a|X) = sup_{\geq k}(X) - sup_{\geq k}(X \setminus \{a\})$$

*Proof.* See Appendix C.

Calculation of this formula can be done concurrently while calculating partial supports, hence the complexity of the algorithm is not increased.

In this experiment, the parameter  $\varepsilon$  is set to 1, and  $min\_sup^{item}$  is set to 0.5 of the minimum support. Here we compare our technique with FT-APRIORI [16], BIAS [17], and VB-FT-MINE [13]. The results are all as expected, and consistent with previous experiments.

## 8 Conclusion

In this paper we introduced the *partial supports* of itemsets, which is the number of transactions containing

*certain number* items of an itemset. We show that by knowing the *partial supports*, we can derive many interestingness measures of itemsets. Furthermore, we prove the existence of recurrence relations between itemsets, and based on that we design efficient algorithms which are similar to those for FIM.

We integrate our recurrence relation to two algorithms for FIM, Apriori and Eclat, and empirically evaluate the efficiency compared to previous techniques. Our experimental results show that our techniques are comparable with normal FIM algorithm, and much more efficient compared to others.

The bottleneck of our algorithms is the need to store all results. While they are stored *by default* in Apriori algorithms, we need to create additional hash-table for Eclat algorithms. This requirement results in the huge space complexity. Therefore, we are interested in designing similar techniques, which are more space efficient, or if possible do not require a hash table.

Another interesting future work is to improve the itemset traversal procedure. Using current technique, we can only mine interesting itemsets that satisfy anti-monotonicity property. We are interested to extend it for the cases that do not satisfy the anti-monotonicity property.

## References

- [1] R. Agrawal, T. Imielinski, and A.N. Swami. Mining association rules between sets of items in large databases. In *Proc. of SIGMOD*, pages 207–216, 1993.
- [2] S. Agrawal, V. Krishnan, and J.R. Haritsa. On Addressing Efficiency Concerns in Privacy-Preserving Mining. In *Proc. of DASFAA*, pages 113–124, 2004.
- [3] C. Borgelt. Efficient Implementations of Apriori and Eclat. In *Proc. of FIMI*, 2003.
- [4] T. Brijs, G. Swinnen, K. Vanhoof, and G. Wets. Using association rules for product assortment decisions: A case study. In *Proc. of SIGKDD*, pages 254–260, 1999.

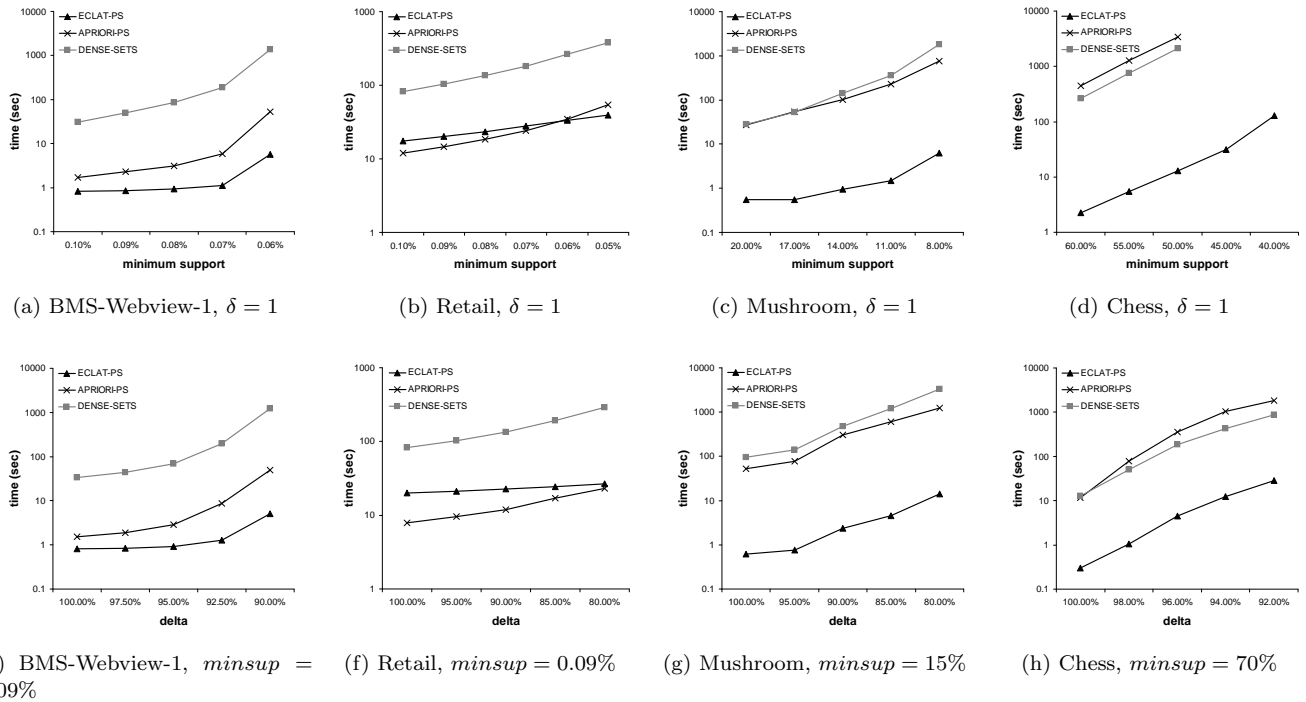


Figure 2: Running time for mining dense itemsets

- [5] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *Proc. of SIGMOD*, pages 265–276, 1997.
- [6] T. Calders and B. Goethals. Mining all non-derivable frequent itemsets. In *Proc. of PKDD*, pages 74–58, 2002.
- [7] T. Calders and B. Goethals. Depth-First non-derivable itemset mining. In *Proc. of SDM*, pages 250–261, 2005.
- [8] T. Calders and B. Goethals. Quick inclusion-exclusion. In *PKDD Workshop Knowledge Discovery in Inductive Databases*, pages 86–103, 2005.
- [9] H. Cheng, P.S. Yu, and J. Han. AC-Close: Efficiently mining approximate closed itemsets by core pattern recovery. In *Proc. of ICDM*, pages 839–844, 2006.
- [10] W. DuMouchel and D. Pregibon. Empirical bayes screening for multi-item associations. In *Proc. of SIGKDD*, pages 67–76, 2001.
- [11] R. Gupta, G. Fang, B. Field, M. Steinbach, and V. Kumar. Quantitative evaluation of approximate frequent pattern mining algorithms. In *Proc. of SIGKDD*, 2008.
- [12] Y. Ke, J. Cheng, and W. Ng. Mining quantitative correlated patterns using an information-theoretic approach to quantitative association rule mining. In *Proc. of ICDE*, pages 227–236, 2006.
- [13] J-L. Koh and P-W. Yo. An efficient approach for mining fault-tolerant frequent patterns based on bit vector representations. In *Proc. of DASFAA*, pages 568–575, 2005.
- [14] J. Liu, S. Paulsen, X. Sun, W. Wang, A. Nobel, and J. Prins. Mining approximate frequent itemsets in the presence of noise: Algorithm and analysis. In *Proc. of SDM*, pages 405–416, 2006.
- [15] E.R. Omiecinski. Alternative interest measures for mining associations in databases. In *IEEE TKDE*, 15(1), pages 57–69, 2003.
- [16] J. Pei, A.K.H. Tung, and J. Han. Fault-tolerant frequent pattern mining: Problems and challenges. In *Proc. of DMKD*, 2001.
- [17] A.K. Poernomo and V. Gopalkrishnan. Mining statistical information of frequent fault-tolerant patterns in transactional databases. In *Proc. of ICDM*, pages 272–281, 2007.
- [18] S. Rizvi and J.R. Haritsa. Maintaining Data Privacy in Association Rule Mining. In *Proc. of VLDB*, pages 682–693, 2002.
- [19] J.K. Seppänen and H. Mannila. Dense itemsets. In *Proc. of SIGKDD*, pages 683–688, 2004.
- [20] N. Tatti. Maximum entropy based significance of itemsets. In *Proc. of ICDM*, pages 312–321, 2007.
- [21] C. Yang, U.M. Fayyad, and P.S. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *Proc. of SIGKDD*, pages 194–203, 2001.
- [22] M.J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New Algorithms for Fast Discovery of Association Rules. In *Proc. of SIGKDD*, pages 283–296, 1997.
- [23] M.J. Zaki and K. Gouda. Fast vertical mining using

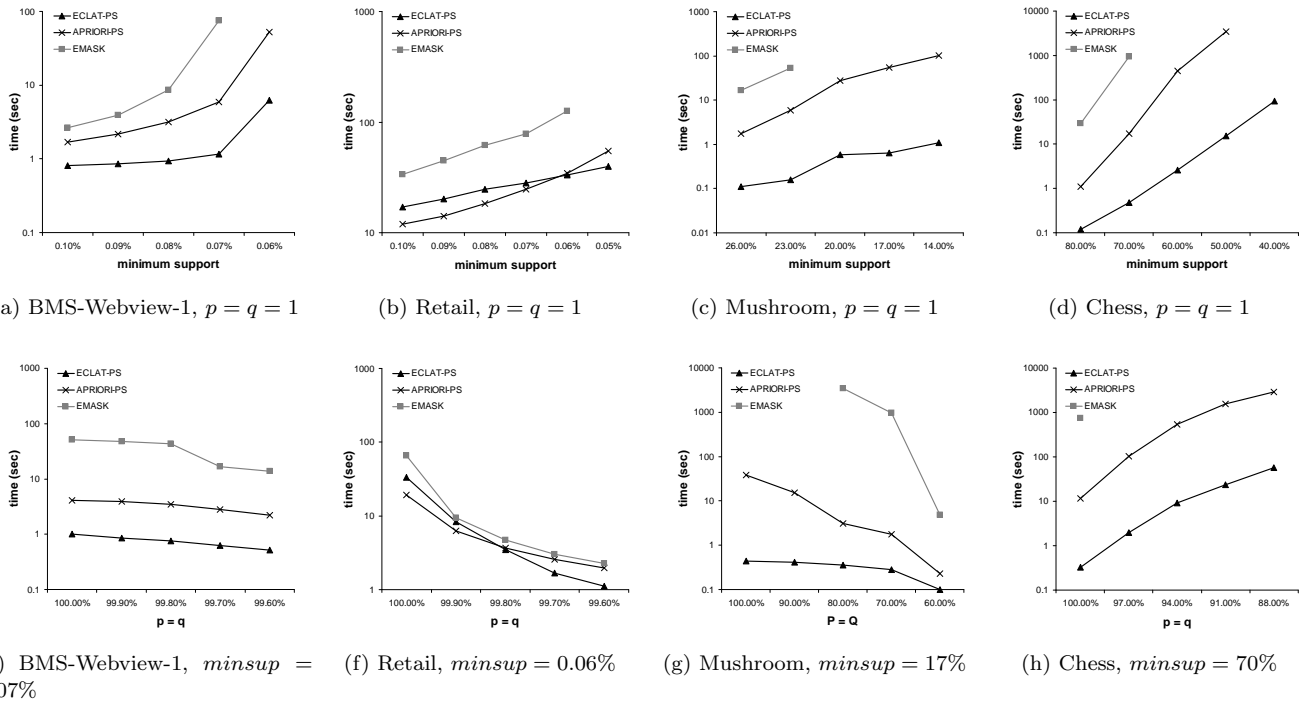


Figure 3: Running time for MASK

diffsets. In *Proc. of SIGKDD*, pages 326–335, 2003.

- [24] Z. Zheng, R. Kohavi, and L. Mason. Real world performance of association rule algorithms. In *Proc. of SIGKDD*, pages 401–406, 2001.

## APPENDIX

### A Proof of Equation (3.1)

The expected support of an itemset after the database is affected with Bernoulli noise can be calculated using matrix operations. Let  $sup'(x)$  be the expected support of item  $x$  after Bernoulli noise, and  $sup(\bar{x})$  be the support of negation of  $x$ , that expected support can be calculated as:

$$\begin{pmatrix} sup'(x) \\ sup(\bar{x}) \end{pmatrix} = M \cdot \begin{pmatrix} sup(x) \\ sup(\bar{x}) \end{pmatrix}$$

where

$$M = \begin{pmatrix} p & 1 - q \\ 1 - p & q \end{pmatrix}$$

To estimate the original support given the noisy database, we can inverse the matrix  $M$ , such as:

$$\begin{pmatrix} sup(x) \\ sup(\bar{x}) \end{pmatrix} = \frac{1}{p + q - 1} \begin{pmatrix} q & q - 1 \\ p - 1 & p \end{pmatrix} \cdot \begin{pmatrix} sup'(x) \\ sup'(\bar{x}) \end{pmatrix}$$

Since the noise is i.i.d, the inverse is also i.i.d. Hence, the expected support of  $k$ -itemsets is simply

the product of  $k$  1-itemset supports. Given the noisy database, the estimated true support of an itemset, before the database was affected by Bernoulli noise can be calculated as

$$\sum_{k=0}^{|X|} sup_{=k}(X) \cdot \left(\frac{q}{p + q - 1}\right)^k \cdot \left(\frac{q - 1}{p + q - 1}\right)^{|X| - k}$$

### B Proof of Equation (5.9)

We prove the recurrence relation using induction. The base case when  $|X| = 1$ , and  $k = |X|$  is trivial. Next, we prove the induction phase.

First, we analyze itemsets subsumed by  $sup_{\geq k}(X \setminus \{a\})$ . Recalling the definition, itemsets included in this function are those containing at least  $k$  items of  $X \setminus \{a\}$ . Hence, considering item  $a$ , a transaction can either (1) contain *at least*  $k + 1$  items of  $X$ , or (2) not contain  $a$  and contains *exactly*  $k$  items of  $X$ . These 2 parts are *disjoint* and *complete*. Note that (1) is equivalent to  $sup_{\geq k+1}(X)$ .

EXAMPLE B.1. Let  $X$  be  $abcd$ . In this example, for clarity, whenever we write transactions containing  $ab$ , it denotes transactions that only contain  $ab$  and do not contain items  $c$  and  $d$ , or equivalently  $ab\bar{c}\bar{d}$ .

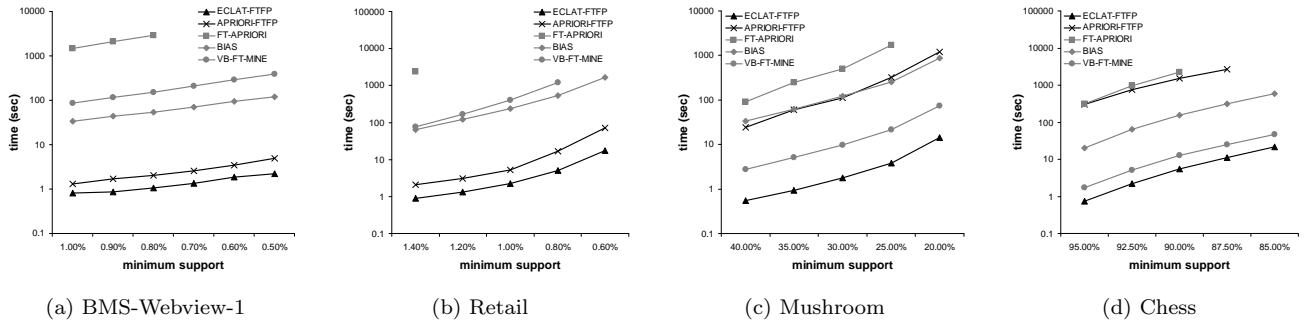


Figure 4: Running time for mining FTFP

$sup_{\geq 2}(bcd)$  counts number of transactions containing itemsets  $bc, bd, cd, bcd, abc, abd, acd,$  and  $abcd$ . We split these itemsets into two parts. The first is the one that consists of more than 2 items, viz.,  $bcd, abc, abd, acd,$  and  $abcd$ . The second is the one consisting of exactly 2 items, but not containing item  $a$ , which is  $bc, bd,$  and  $cd$ . Note that these two parts are disjoint and complete.

Since we sum those terms for all items  $a \in X$ , at the end we have (3)  $|X|$  multiplies of  $sup_{\geq k+1}(X)$ , and (4) summation of transactions containing exactly  $k$  items, each not containing one item of  $X$ . Now we count how many times each  $k$ -subset of  $X$  appears in (4). Each  $k$ -subset of  $X$  contains  $k$  items, and does not contain  $|X| - k$  items. Hence, it appears exactly  $|X| - k$  times in (4). Equating the numerators of equation 5.9, we have  $|X| - k$  multiple of itemsets containing exactly  $k$  items, and also  $|X| - k$  multiple of itemsets containing at least  $k + 1$  items, hence together we have  $|X| - k$  multiple of itemsets containing at least  $k$  items, which is  $(|X| - k) \cdot sup_{\geq k}(X)$ .

To attain  $sup_{\geq k}(X)$ , we simply divide the result by  $|X| - k$ . ■

### C Proof of Equation (7.11)

As mentioned in proof of Equation 5.9 (Appendix B),  $sup_{\geq k}(X \setminus \{a\})$  counts transactions that either (1) contains at least  $k + 1$  items of  $X$  which is  $sup_{\geq k+1}(X)$ , and (2) not contains  $a$  and contains exactly  $k$  items of  $X$ . Similarly, we can split  $sup_{\geq k}(X)$ , into  $sup_{=k}(X) + sup_{\geq k+1}(X)$ .

Therefore, subtracting  $sup_{\geq k}(X \setminus \{a\})$  from  $sup_{\geq k}(X)$  gives transactions that contains  $a$ , and contains exactly  $k$  items of  $X$ , which is our target. ■