Association against Dissociation: some pragmatic considerations for Frequent Itemset generation under Fixed and Variable Thresholds

Sukomal Pal
Indian Statistical Institute
203, B.T.Road, Kolkata, India
Sukomal_r at isical.ac.in

Aditya Bagchi
Indian Statistical Institute
203, B.T.Road, Kolkata, India
aditya at isical.ac.in

ABSTRACT

Traditionally, *support* is considered to be the standard measure for frequent itemset generation in Association Rule mining. This paper provides a new measure called *togetherness* where dissociation among items is also considered as a parameter in the frequent itemset generation process. Results of performance analysis show that association against dissociation is a more pragmatic approach and discovers truly associated candidate itemsets. Second part of the paper extends this *togetherness* measure to the domain of variable threshold. Here, like variable minimum *support*, a variable minimum *togetherness* has been proposed where this minimum value decreases as the itemset size increases. A simple and pragmatic process has been described, which can be easily implemented. It also provides ample control facilities in the hand of the users. Necessary change and extension of the existing algorithms have been made to establish the concepts. Here as well, results of performance analysis justify the approach.

Keywords

Association rule mining, frequent itemset, support, togetherness.

1. INTRODUCTION

Discovery of association rules is an area of study in data mining. Starting from the earliest work in this area[1], many efficient methods including parallel algorithms have been developed[2,5,6,7]. The two usual metrics for measuring association among different items are *support* and *confidence*. Considering a set of transactions containing different data items, an association between data itemset X and data itemset Y, represented as $X \Rightarrow Y$, signifies that the transactions that contain X tend to contain Y as well. In this context, the first measure of association called *support* is obtained as,

"the support for a set of items is the % of transactions that contain all of these items".

In other words, if A and B are any two items, then P(AB) or the joint probability of A and B is its support. A set of items will be considered for mining rules if its support is above a threshold called *MINSUP*. A user, interested in mining a data set, normally specifies this minimum support or *MINSUP*. An itemset that crosses the *MINSUP* threshold is called a frequent itemset.

Similarly, the measure of *confidence* is obtained as,

"out of the transactions that have the itemset LHS, the % of transactions that have RHS as well, is the measure of confidence of the rule LHS $\Rightarrow$ RHS".

In other words, between two items A and B, it is the probability of the availability of one given the other, i.e. P(B/A) or the conditional probability is the measure of *confidence* for the rule $A \Rightarrow B$.

This type of association is called binary association, where in each transaction, only the presence and absence of the items are considered. So, an environment involving *n* data items, would tend to produce 2^n possible itemsets for which *support* has to be measured. However, if a particular itemset fails to cross the *MINSUP* threshold or does not become a frequent itemset, all its supersets would not cross the *MINSUP* threshold as well. So, the frequent itemset generation algorithm will not generate the supersets of an itemset X, if X itself fails to cross the *MINSUP* threshold. Starting from the well known Apriori algorithm[2], almost all frequent itemset generation algorithms follow the same principle.

However, for an itemset X, a transaction is considered for counting support only if all the items in the itemset are present in the transaction. Figure-1 shows 10 transactions involving 4 items. Here presence of an item is denoted by 1 and absence by 0. TID is the transaction-id and the data set has 4 items A,B,C and D.

TID	A	B	C	D
1	1	1	0	0
2	0	0	1	0
3	1	1	1	1
4	1	0	0	0
5	0	1	0	1
6	1	1	0	0
7	0	1	1	1
8	1	0	1	1
9	1	1	0	0
10	1	0	1	1

Figure 1: An example set of ten transactions.

Considering *MINSUP* =0.3, for itemsets AB and CD (i.e. 30% of transactions should have AB or CD appearing together),

support(AB) = 0.4 and *support*(CD) = 0.4.

Both the itemsets cross the *MINSUP* threshold and they have the same support. So, both AB and CD are frequent itemsets and should be considered for subsequent measure of *confidence*.

However, the dataset also provides the count where out of the two items in the example itemsets, only one is absent. In other words, for itemsets AB or CD, it provides the count of occurrences of 10 and 01 patterns. If 11 pattern provides a measure of association, 10 or 01 pattern should provide a measure of dissociation. Considering the same set of itemsets again,

dissociation(AB) = 0.5 and *dissociation*(CD) = 0.2.

So, though both the itemsets AB and CD had the same *support*, if the measure of dissociation is considered, CD has less dissociation than AB. So, the dissociation of an itemset is obtained by finding,

the % of transactions where one or more items but not all are absent.

Logically speaking, between two itemsets of same size (e.g. both AB and CD are 2-itemsets) and same *support*, the one having less dissociation should be considered to have stronger association. This paper provides a measure of *togetherness* similar to *support*, for extracting frequent itemsets from a set of transactions under high association but low dissociation. The algorithm is similar to the well known Apriori algorithm. Authors are aware that Apriori is not a very efficient algorithm and many better algorithms for frequent itemset generation have already been developed. However, since the purpose of the paper is to establish a new idea of association, efficiency of the algorithm has taken a back seat here. Future efforts would try to develop better algorithms.

In the present paper, adequate test results have been provided to compare the usual *support* measure against the new measure of *togetherness*. It has been found that measure of *togetherness* tends to discover more frequent itemsets than that is done under the usual *support* measure. The results have been explained and justified.

An earlier effort[3] has provided a similar measure called *similarity*. However, the authors did not observe the possibility of defining a stronger association minimizing dissociation among items. On the other hand the authors considered the *similarity* as a measure in lieu of *support* in the environment where *confidence* is high but *support* is low. In the process, the authors studied the phenomenon for two itemsets only and observed that the measure of *similarity* for higher orders of itemsets is computationally prohibitive. The present paper considers *togetherness* for any k-itemsets and provides a simple method for its enumeration.

Another recent effort[6] has considered the concept of variable *MINSUP* constraint. The classical way of frequent itemset generation considers uniform value of *MINSUP* irrespective of the size of the itemset. However, this is quite logical that if itemset AB has a certain *support*, the *support* of its supersets would tend to decrease as the size of the superset increases. So, it would definitely be desirable if the value of *MINSUP* becomes a function of the itmesize and its value decreases as the size of the itemset increases. Therefore, a method should be developed where the *MINSUP* defined by a user should get modified in such a way that the *MINSUP* for (k+1)-itemset is less than that for k-itemset. In [6], an elaborate method has been proposed, first to classify the items, then to define *MINSUP* ranges for each class and then again, a method to change the *MINSUP* within a range as the itemset size increases. Accordingly, the authors have defined an Adaptive Apriori algorithm and studied its performance against the classic Apriori algorithm.

As mentioned earlier, this paper considers *togetherness* as the effective measure for frequent itemset generation in lieu of *support* and user may specify a *min_togetherness* similar to *MINSUP*. Extending this idea to an environment similar to variable *MINSUP* proposed by [6], this paper offers a very simple and pragmatic method to specify variable *min_togetherness* where this threshold value decreases as the itemset size increases. However in the proposed method, a user can specify a maximum value of the threshold and an acceptable lower bound. As a result, even when the threshold is decreased with the increase of the itemset size, it never goes below the lower bound irrespective of the extent of increase in the itemset size. Experimental results have been provided to elaborate the process.

While Section 1 provides the introduction Section 2 covers the effect of dissociation in frequent itemset generation. Section 3 discusses the variable threshold phenomenon and Section 4 draws the conclusion.

2. THE DISSOCIATION EFFECT

2.1 Togetherness in lieu of Support

An itemset is accepted as a frequent itemset if its *support* crosses the *MINSUP* threshold.

Let T= Total no.of transactions.

Let S_i = The subset of transactions containing the item i.

Let N_i = The no. of occurrences of item i.

= The no. transactions where the item i has appeared.

So, $N_i = |S_i|$ = The cardinality of S_i.

So, the *support* for i = (N_i / T)

Hence, the *support* for an itemset (AB)=The *support* for the set of items $\{A \cup B\} = (N_{AB} / T)$

where, $N_{AB} = |S_A \cap S_B|$

Referring to Figure-1 again, $N_{AB} = N_{CD} = 4$, and hence both the itemsets have *support* = 0.4.

Once again in Figure-1, out of the 10 transactions, 5 of them have either A or B appearing alone. In case of CD, C or D has appeared alone in only 2 transactions. Defining this as the *dissociation* among items, CD is found to have a stronger association than AB even when they have the same *support* value. So, in case of ideal association, all the items of an itemset would either appear together in a transaction or none of them should be present.

Following the classical work of Agrawal and Srikant[2], the problem may be formally presented as,

Let $I = \{A_1, A_2, A_3, .., A_m\}$ be the set of all items in a database where each item A_i is a *boolean* attribute.

Let D be the set of all transactions in the database, where each transaction $T = \{i_1, i_2, \ldots\ldots, i_k\}$ $(k \leq m)$ is a set of items such that T is a subset of I.

So, the whole database may be viewed as a 0/1 matrix of size (|D| x m) with A_i's as columns and transactions as rows. If we take C_i as the set of rows that have 1 in column A_i then the classical measure of *support* for itemset $\{A_i, A_j\} = |C_i \cap C_j| / |D|$.

Now, considering the degree of dissociation along with association, a new measure *togetherness* may be defined as,

togetherness of itemset $\{A_i, A_j\} = |C_i \cap C_j| / |C_i \cup C_j|$.

So, extending the definition to k-itemsets,

togetherness of k-itemset$\{A_1, A_2,\ldots,A_k\}$

$= |C_1 \cap C_2 \cap \ldots \cap C_k| / |C_1 \cup C_2 \cup \ldots \cup C_k|$

Now, for a k-itemset $\{A_1, A_2,\ldots,A_k\}$,

$|C_1 \cap C_2 \cap \ldots \cap C_k|$ = cardinality of the set of rows where all the items $A_1, A_2,\ldots,A_k$ are 1 simultaneously.

Similarly, $|C_1 \cup C_2 \cup \ldots \cup C_k| - |C_1 \cap C_2 \cap \ldots \cap C_k|$ = *dissociation* of itemset $\{A_1,A_2,\ldots A_k\}$ = cardinality of the set of rows where at least one of the items in $\{A_1,A_2,\ldots A_k\}$ is present but not all of them simultaneously.

Some desirable properties of togetherness are:

Lemma 1: F*or any value of k(1 ≤ k ≤ m), togetherness value for k-itemset lies between 0 and 1 (m is the total number of items present in the database).*

support measure for k-itemset is defined as,

support$(A_1,A_2,\ldots A_k) = P(A_1A_2,\ldots A_k)$

= Joint probability of occurrence of all the k data items.

So, the value of *support* is always bounded between 0 and 1. Since *togetherness* has been used in lieu of *support*, it should also exhibit the same property.

Proof :

By definition,

togetherness$(Ai_1, Ai_2,\ldots\ldots, Ai_k) = |Ci_1 \cap Ci_2 \cap \ldots \cap Ci_k| / |Ci_1 \cup Ci_2 \cup \ldots\ldots \cup Ci_k|$

- Since, both the numerator and the denominator are positive integers and the numerator is a subset of the denominator, the ratio providing the *togetherness* measure can never exceed 1.
- The value of *togetherness* will be equal to 1 only if there is no dissociation among the items. In other words, in any transaction, either all the k-items are present or all of them are absent. So, when k =1 i.e. for any 1-itemset $\{Ai\}$, *togetherness*$(Ai) = |Ci| / |Ci| = 1$.
- The value of *togetherness* will be 0, only if $|Ci_1 \cap Ci_2 \cap \ldots \cap Ci_k| = 0$. In other words, there is no transaction where all k-items are present.

So, togetherness measure is bounded between 0 and 1.

Lemma 2: *Togetherness value for k-itemset gradually decreases as k increases in 1≤ k ≤ m or togetherness for (k+1)-itemset ≤ togetherness for k-itemset.*

Proof :

This property is also exhibited by the *support* measure.

The Lemma can be proved by induction.

- For any 2-itemset $\{Ai_1,Ai_2\}$, *togetherness*$(Ai_1,Ai_2) = |Ci_1 \cap Ci_2| / |Ci_1 \cup Ci_2| \le 1$ [from **Lemma 1**].
- For any 3-itemset $\{Ai_1,Ai_2,Ai_3\}$,

  $(Ci_1 \cap Ci_2 \cap Ci_3)$ is a subset of any of $(Ci_1 \cap Ci_2)$, $(Ci_1 \cap Ci_3)$ or $(Ci_2 \cap Ci_3)$. So,

  $|Ci_1 \cap Ci_2 \cap Ci_3| \le \min\{|Ci_1 \cap Ci_2|, |Ci_1 \cap Ci_3|, |Ci_2 \cap Ci_3|\}$
- Again, each of $(Ci_1 \cup Ci_2)$, $(Ci_1 \cup Ci_3)$ and $(Ci_2 \cup Ci_3)$ is a subset of $(Ci_1 \cup Ci_2 \cup Ci_3)$. So,

  $|Ci_1 \cup Ci_2 \cup Ci_3| \ge \max\{|Ci_1 \cup Ci_2|, |Ci_1 \cup Ci_3|, |Ci_2 \cup Ci_3|\}$
- So, $|Ci_1 \cap Ci_2 \cap Ci_3| / |Ci_1 \cup Ci_2 \cup Ci_3| \le \min\{|Ci_1 \cap Ci_2| / |Ci_1 \cup Ci_2|, |Ci_1 \cap Ci_3| / |Ci_1 \cup Ci_3|, |Ci_2 \cap Ci_3| / |Ci_2 \cup Ci_3|\}$.

So, *togetherness* for any 3-itemset ≤ *togetherness* for any 2-itemset generated out of those three items.

Proceeding the same way, it can be shown that for any k-itemset,

togetherness for (k+1)-itemset ≤ *togetherness* for k-itemset.

2.2 Enumeration of Togetherness

The numerator of *togetherness* measure is same as the *support* measure. For any k-itemset $\{A_1,A_2,\ldots A_k\}$, it is the count of the transactions where all the k items are present or the all 1's cases. The denominator of *support* measure is the total number of transactions present in the dataset or |D|, as defined earlier. However, in case of *togetherness* measure, enumeration of denominator is more difficult, as it provides the count of the transactions where any or all of the items are present. So apparently, when *support* measure for any k-itemset needs only one pass of the database, enumeration of *togetherness* may require several such passes. The same observation has been made in [3], when they provided the measure of *similarity.* This enumeration process, however, may be simplified in the following way,

From the simple set algebra it can be observed that,

$D = (C_i \cup C_j \ldots\ldots\ldots \cup C_k) \cup (C_i \cup C_j \ldots\ldots\ldots \cup C_k)^c$
where, D is the universal set here and,
C_i^c is the set complement of C_i.
Now applying De Morgan's theorem,
$D = (C_i \cup C_j \ldots\ldots\ldots \cup C_k) \cup (C_i^c \cap C_j^c \ldots\ldots\ldots\ldots \cap C_k^c)$
Now as the two sets are disjoint,
$|D| = |C_i \cup C_j \ldots\ldots\ldots \cup C_k| + |C_i^c \cap C_j^c \ldots\ldots\ldots\ldots \cap C_k^c|$
So, $|C_i \cup C_j \ldots\ldots\ldots \cup C_k| = |D| - |C_i^c \cap C_j^c \ldots\ldots\ldots\ldots \cap C_k^c|$

Hence, the effort of enumerating the denominator of *togetherness* measure turns into the process of enumerating the set $(C_i^c \cap C_j^c \ldots\ldots\ldots\ldots \cap C_k^c)$. This set is actually the all-0 set, as C_i^c denotes the set of transactions where item A_i does not occur or simply the column entry is 0. Following the same process as the Apriori algorithm, the all-0 set can be counted in O(|D|) time by a single pass over the entire database, which is definitely a substantial reduction in overall complexity. Again a careful observation shows that counting of this all-0 set need not be done after the counting of all-1 set (numerator of *support* or *togetherness* measure). In fact, both enumerations can be done in the same pass over the database. Thus the complexity of enumerating *togetherness* is of the same order as *support*. As a matter of fact, against each disk access, time required for the computation of *togetherness* will be slightly higher than that required for the computation of *support*. This small difference is due to the main memory processing time needed for counting the all 0 transactions (i.e. transactions where all the k items are absent) in addition to all 1 transactions.

2.3 Aprioridis Algorithm

Classical Apriori algorithm is the most well known method to generate frequent itemsets against a pre-specified minimum

support, *MINSUP*. Though many efficient algorithms have been developed since Apriori was proposed, the principle of frequent itemset generation has remained the same. In order to investigate the possibility of finding a new measure like *togetherness* in lieu of *support*, an Aprioridis algorithm, similar to Apriori, has been developed. It generates frequent itemsets against a pre-specified minimum threshold of *togetherness* or *min_togetherness*. Since the value of *togetherness* is 1 for all 1-itemsts [**Lemma 1**], the Aprioridis algorithm starts generating the frequent itemsets from 2-itemset onwards. As observed in [3], this approach would ensure that some items are not discarded in the first pass itself because of their small *support*, even when they exhibit strong association later when taken with other items to form higher order itemsets. This is verified by subsequent *confidence* measure in [3].

For discovering frequent itemsets, Apriori algorithm makes multiple passes over the entire data set. So, generation of frequent k-itemsets needs k passes. The frequent itemsets generated in one pass are used as the seeds for the subsequent pass. This process continues till no frequent itemset is found for a particular value of k. As mentioned earlier, an itemset is said to be frequent if it exceeds the *MINSUP* threshold. Also, in case of standard *support* metric, only presence of all k-items or all 1's count contributes towards the *support* measure.

D	Total data set, given as a 1-0 matrix.
T	A transaction in D.
L_k	Set of frequent k-itemsets. Each member has 3 fields; itemset, 1's count & togetherness value.
C_k	Set of candidate k-itemsets that are potentially frequent itemsets. Each member has 3 fields similar to L_k.
$L_k[r]$	r-th itemset in any set of frequent itemsets L_k.
O(r)	1's count for itemset r (all items of itemset r are present).
Z(r)	0's count for itemset r (none of the items of itemset r is present).
min_togetherness	Pre-specified minimum value of togetherness.

Figure 2: Notation used in Aprioridis Algorithm

The Aprioridis algorithm follows the same method, though the first pass is avoided. All 1-itemsets are considered to be frequent. Moreover in each pass, for k-itemsets, both all 1's (all k items are present) and all 0's (none of the k-items is present) counts are taken. While the all 1's count forms the numerator, all 0's count contributes to the denominator of the *togetherness* measure.

In the algorithm, all members of frequent k-itemsets L_k satisfy the minimum threshold requirement. Each pass in the algorithm consists of two phases. First, the candidate-set C_k is generated from the frequent-sets L_{k-1} of the previous pass using **candi-gen** function. Next, the database is scanned for each member of the candidate-set C_k to find its *togetherness* value and frequent k-itemset L_k is determined using the function **prune**. Notations used in the algorithm are listed in Figure-2.

The Aprioridis algorithm is given as :

```
Main Program: Aprioridis
L1 = {large 1-itemsets};
for (k=2; Lk-1 ≠ 0; k++) do
begin
    Ck = candi-gen (Lk-1);
    Lk = prune (Ck, D, min_togetherness);
end
Answer = ∪k Lk;
```

Here, **Answer** provides the union of all the frequent itemsets produced between 1 to k-itemsets. These are the frequent itemsets to be considered for computation of *confidence* and subsequent association rule generation.

```
function candi-gen(Lk-1)
begin
    Ck := Φ;
    for all Lk-1[r], Lk-1[j] in Lk-1
    with Lk-1[r] = { i1,…ik-2,ik-1}
    and Lk-1[j] = { i1,…ik-2,i'k-1} do
    /* where ik-1 ≠ i'k-1 */
    begin
        f : = Lk-1[r] ∪ Lk-1[j] = { i1,…ik-2,ik-1,i'k-1}
        if for all item i in f , {f – i} belongs to Lk-1
        then Ck := Ck ∪ f ;
    end
    return Ck ;
end
```

The candi-gen function takes as argument L_{k-1}, the set of all frequent (k-1)-itemsets and returns C_k, all candidate k-itemsets.

This candi-gen function is exactly similar to the join step executed in the Apriori algorithm. Each pair of itemsets belonging to L_{k-1} (say, $L_{k-1}[r]$ and $L_{k-1}[j]$) are compared and if they match in first (k-2) terms, then a candidate k-itemset is generated taking the matched (k-2) terms and adding the (k-1)th. term of $L_{k-1}[r]$ and $L_{k-1}[j]$. Thus the set of candidate k-itemsets C_k is generated.

```
function prune(C, D, min_togetherness)
/* C is the set of candidate itemsets and
D is the total data set */
begin
    for all itemset c in C do
    begin
        O(c) = 0;
        Z(c) = 0;
    end
    for all c in C do
    begin
        for all t in D do
        begin
            if c is in t and it has all 1's then
            O(c) = O(c) +1;
            else if c is in t and it has all 0's then
            Z(c) = Z(c) + 1;
        end
        togetherness(c) = O(c) / (|D| - Z(c));
    end  /*  candidate set ends  */
    L :=0;
    for all c in C do
    begin
        if togetherness(c) ≥ min_togetherness
        then L := L U c;
    end
    return L;
    /* L provides the frequent itemsets */
end
```

2.4 Performance Evaluation

To assess the performance of the Aprioridis algorithm against the classic Apriori algorithm, experiments have been made using a set of synthetic data. Both the algorithms have been implemented using C language and all the experiments have been conducted on a SUN Enterprise sever under Solaris 9.0 environment. The transactions in the experiment mimic the retailing environment. Number of items taken for the experiment is 120 and the number of transactions has been increased gradually from 5000 to 20000 in steps of 5000. For the purpose of demonstrating the relative performance, the number of 3-itemsets generated by the two algorithms at two different threshold values are shown in Figure-3(a) and 3(b). In both the cases the *MINSUP* and *min_togetherness* have been made equal.

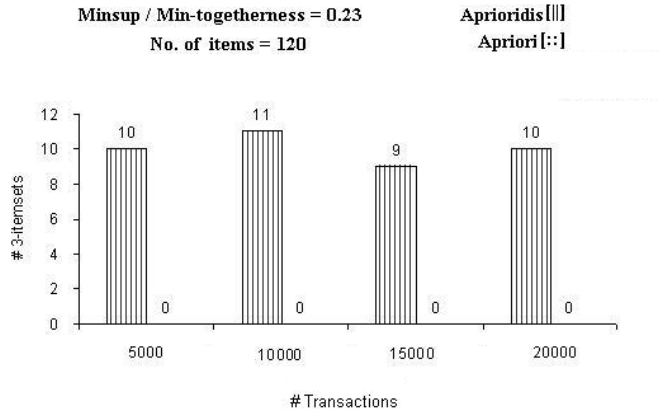

Figure 3: Performance of Aprioridis and Apriori algorithms with threshold = 0.23

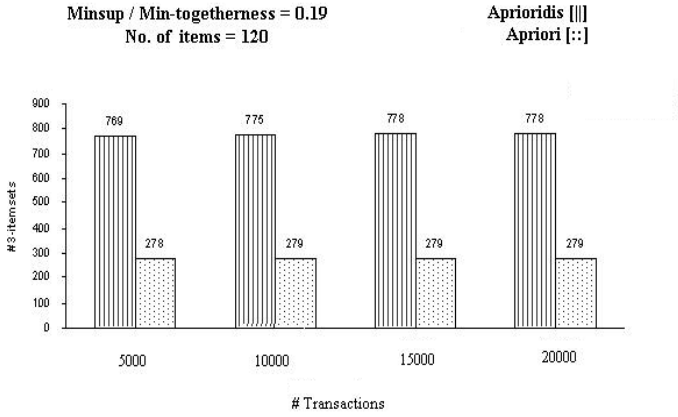

Figure 4: Performance of Aprioridis and Apriori algorithms with threshold = 0.19

The results shown in Figure-3 and Figure-4 are apparently trivial. The expressions of *support* and *togetherness* have the same numerator, i.e. $|C_i \cup C_j \ldots\ldots\ldots \cup C_k|$. On the other hand, the denominator of *support*, i.e. |D|, is greater than the denominator of *togetherness*, i.e. $|D| - |C_i^c \cap C_j^c \ldots\ldots\ldots\ldots \cap C_k^c|$. So, it is obvious that for the same threshold value, more number of itemsets would cross the threshold in case of *togetherness* measure than in case of *support* measure. However, some important observations can still be made from these results.

Figure-3 shows the case where in spite of strong association among items, the *support* measure fails to reveal them just because the dissociation among the items has not been considered as a part of the process of discovering association. Incidentally, same observation was made by [3], however they attributed it to high confidence under low support. As a matter of fact, Figure-3 justifies the inclusion of dissociation as a parameter in the discovery of association among items.

2.4.1 Efficiency in Association Rule Generation

Figure-4 shows the extent of loss of information (the possible association among items), if a user prefers to specify the same threshold value for *min_togetherness* as well as *MINSUP*. Irrespective of the number of transactions, the number of 3-itemsets chosen against *togetherness* measure is more than 2.7 times the same revealed by *support* measure. Another important observation can again be made from this result.

The expression of *togetherness* in Section 2.2 shows that if for any k-itemset the all 0's count is 0, i.e. the case of maximum dissociation, the value of *support* measure will be equal to the value of the *togetherness* measure. Otherwise, for the same k-itemset, the *support* value will always be less than the *togetherness* value. So, an itemset that crosses the *MINSUP* threshold would definitely cross the *min_togetherness* threshold if both the thresholds are made equal. This becomes apparent from Figure-4. So, it is absolutely justified if the *min_togetherness* threshold is set to a higher value than the *MINSUP* threshold for the same data set and the same k-itemset.

Going back to the data set shown in Figure-1, both the two itemsets AB and CD have the *support* value of 0.4. If *MINSUP* is 0.3, both the itemsets become frequent itemsets, even when AB has a much higher *dissociation* than CD. Now, if the *min_togetherness* value is set to 0.5, a considerably higher value than *MINSUP*, CD crosses the threshold (*togetherness* = 0.67), but AB fails (*togetherness* = 0.44) to do so. In other words, if *togetherness* measure is considered in lieu of *support*, the itemsets having considerably high dissociation will not become frequent itemsets.

Now, for the generation of a rule like $X \Rightarrow Y$, the *confidence*(XY) = N(XY) / N(X) must cross the *min_confidence* value specified by the user. If *togetherness* measure is used in lieu of *support*, *min_togetherness* threshold will ensure that XY will have low dissociation in order to become a frequent itemset. So, as such, this measure would ensure that N(XY) is sufficiently close to N(X). Once again referring back to Figure-1,

For a rule $A \Rightarrow B$, *confidence*(AB) = N(AB) / N(A) = 0.57,

and for a rule $C \Rightarrow D$, *confidence*(CD) = N(CD) / N(C) = 0.8

Now, if *min_confidence* is set to 0.7, rule $C \Rightarrow D$ will be accepted but not $A \Rightarrow B$. So, even when both the 2-itemsets had the same *support*, the one with less dissociation could form the rule. However, a suitable choice of *min_togetherness* already discarded the itemest AB as a frequent itemset. So, instead of *MINSUP,* a suitable choice of *min_togetherness* would discard many of the frequent itemsets that would have high *support* but low *confidence* because of high *dissociation* among the participating items. Effectively, the rule discovery process would generate less number of frequent itemsets that would be discarded during testing of *confidence*. Thus the rule generation process would be more efficient.

3. VARIABLE TOGETHERNESS

The classical way of frequent itemset generation considers uniform value of *MINSUP* irrespective of the size of itemset. Starting from the Apriori algorithm, other algorithms proposed later are, no doubt, computationally more efficient but still follow the same principle of uniform *MINSUP*. However, this is quite logical to consider that if itemset AB has a certain *support*, the *support* of any of its supersets (say ABC) will be less than or, at most, equal to the *support* of AB. So, starting from 1-itemset, the *support* would tend to decrease as the size of the itemset increases. So, the user specified *MINSUP* value should also be decreased gradually with the increase of the itemset size. Without this principle, the frequent itemset generation process may fail to include many desirable higher order itemsets and consequently may not generate many desirable higher order association rules. A trivial solution to this problem will be to set a very low *MINSUP* value so that higher order itemsets can also cross the threshold. However, this process would unnecessarily create too many lower order itemsets that would get discarded later by the *confidence* measure.

So, it would definitely be desirable if the value of *MINSUP* also decreases as the size of the itemset increases. As a matter of fact, the *MINSUP* value should be so specified that it becomes a monotonically decreasing function of itemeset size, i.e. the value of *MINSUP* should decrease as the itemset size increases.

A recent effort[6] has considered the concept of variable *MINSUP* constraint. Here the authors have proposed a method for specifying variable *MINSUP* that gets modified as the itemsize increases. Accordingly, the authors have defined an Adaptive Apriori algorithm. The salient features of the process are:

a) The items are separated in different bins where the items in one bin would have a particular *MINSUP* value. Different bins may have different values of *MINSUP*.

b) Next is the use of support constraint to generate dependency chain of itremsets, so as to create schema enumeration tree where each node, except the root, is labeled by a bin and a range for *MINSUP* is also specified.

c) Depending on the application, the system determines the minimum *support* in different ways. It can be support-based spcification, where the bins are formed by computing the support of individual items in one pass of the transactions and then clustering the items based on their supports. The *MINSUP* of a bin may be specified as the maximum, minimum or the average support of that bin. Other ways may be concept-based specification or attribute-based specification or something else.

d) In Apriori algorithm, a candidate k-itemset is usually generated by the combination of two (k-1)-itemsets where both such sets are frequent. The participating (k-1)-itemsets and the generated k-itemset have the same *MINSUP*. However, it is not true in case of Adaptive Apriori. The minimum *support* for the three itemsets may be different. The authors in [6] have taken an approach that replaces *MINSUP* with a new function *Pminsup*, called the *pushed minimum support* such that *Pminsup* defines a superset of the frequent itemsets and this superset can be computed in the manner of Apriori. The paper goes on defining the different properties of *Pminsup*. It also proposes a method of specifying the minimum *support* as a function of the product of k supports. Since a *support* basically signifies a probability value i.e. a value less than 1, a product of k supports creates a very low value. So, the product is

multiplied by a factor γ^{k-1}. Thus the minimum *support* takes the form,

$MINSUP = \min \{\gamma^{k-1}(P_1 X\ P_2 X \ldots X\ P_k), 1\}$

Where, any P_i represents a probability value i.e. a value less than 1 and γ is an integer greater than 1.

Now varying the value of γ, the authors have studied the performance of their Adaptive Apriori algorithm against the classic Apriori algorithm using a synthetic dataset, where γ is varied from 1 to 20 and maximum value of k is 7.

The detail treatment of the method is given in [6]. In spite of the fact that the method given by the authors are quite elaborate, depending on the application, the method may ask for considerable domain knowledge from the users, particularly for concept-based or attribute-based specifications.

Extending the idea of using *togetherness* in lieu of *support*, this paper proposes an environment, where, similar to variable *MINSUP* in [6], a *min_togetherness* value may be specified by the user. This threshold value is basically the maximum value of the threshold applicable to the minimum itemset size, i.e. 2-itemsets, and the *min_togetherness* decreases as the itemset size increases. The method is very simple and the approach is very pragmatic. The *min_togetherness* is defined as a monotonically decreasing function of the itemset size. As a result, as the itemset size increases, the *min_togetherness* decreases. Depending on the function used, minimum value of the threshold is always restricted to a lower bound of the original *min_togetherness* value specified by the user. At the same time, the user can specify not only the maximum value of the threshold, he/she can also specify the required function of itemsize k. This control over the specification of the function allows the user to restrict the lower bound of the threshold. So, the *min_togetherness* here, takes the nature,

min_togetherness = f(k). *mt*

where, f(k) is a monotonically decreasing function of itemset size k and *mt* is the maximum value of the threshold specified by the user and applicable to minimum itemset size.

3.1 Variable Aprioridis Algorithm

Main Program: Aprioridis
L_1 = {large 1-itemsets};
for (k=2; $L_{k-1} \neq 0$; k++) do
begin
  C_k = candi-gen (L_{k-1});
  min_togetherness = vth(mt,k);
  L_k = prune (C_k, D, min_togetherness);
end
Answer = $\cup_k L_k$;

The Variable_Aprioridis algorithm, shown above, is almost similar to the one given in section 2.3. Same **candi-gen** and **prune** functions are used. Only a user specified function **vth** has been defined that controls the variation and the lower bound of the maximum threshold **mt**, once again supplied by the user.

3.2 Performance under Variable Threshold

Using the same synthetic data set, experiments have been performed to study the relative performance of the Aprioridis algorithm under fixed and variable thresholds. Here, the **vth** function is so used that the maximum threshold specified is bounded between **mt** and **0.5mt**. As shown in the proof of **Lemma-1** in section 2.1, in case of *togetherness* measure, all 1-itemsets are considered to be frequent. So, the minimum size of the itemset considered for pruning is 2. So, the *min_togetherness* threshold will be equal to **mt** for 2-itemsets. Now, as the itemset size k increases and tends to infinity, *min_togetherness* value tends to **0.5mt**.

```
function vth(mt,k)
  begin
      min_togetherness = (k / (2*(k-1))) * mt
      return min_togetherness;
  end
```

The function **vth** used for the performance study is very simple. Since,

$$\lim_{k \to \infty} k / (2 (k-1)) = 1/2$$

the value of *min_togetherness* remains bounded between **mt** and **0.5mt**. Changing the monotonically decreasing function **vth**, a user can define any lower bound within which the user specified *min_togetherness* value should be bounded irrespective of the value of the itemsize k.

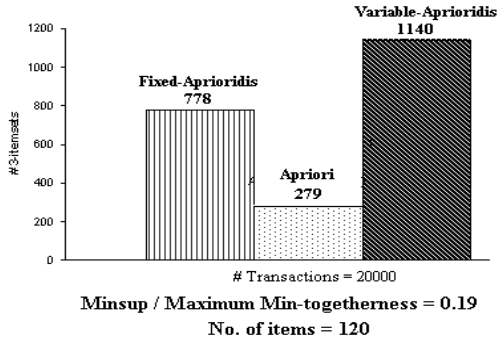

Figure 5: Performance of Fixed and Variable Aprioridis Algorithms for 3-itemsets.

Figure-5 shows the performance of the Aprioridis algorithm under fixed and variable *min_togetherness*. The study has used the same synthetic data set against which the results are shown in Figure-3 and Figure-4. To compare the results with that shown in Figure-4, this study has considered number of transactions as 20000 and a *MINSUP* or maximum value of *min_togetherness* as 0.19. In Figure-5, the number of frequent 3-itemsets generated under Variable_Aprioridis is compared against those generated under

Apriori and Aprioridis (with fixed *min_togetherness*). From Figure-5 it is apparent that variation in *min_togetherness* with itemset size definitely captures more number of frequent itemsets.

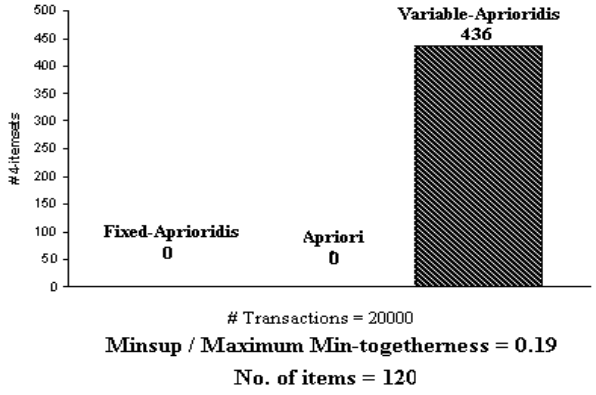

Figure 6: Performance of Fixed and Variable Aprioridis Algorithms for 4-itemsets.

Figure-6 shows more interesting results. With the same threshold, same number of items and same number of transactions, the number of frequent 4-itemsets generated under Variable_Aprioridis is compared against those generated under Apriori and Aprioridis (with fixed *min_togetherness*). It has been found that while both Apriori and Aprioridis (with fixed *min_togetherness*), failed to generate any frequent itemset of size 4, variation in *min_togetherness* could generate a good number of potential associations.

Thus, the study of using variable *min_togetherness* instead of a fixed value has shown that this approach has the potential of generating many extra higher order frequent itemsets that are rejected in the fixed threshold approach. Consequently this approach would also provide more number of interesting associations among items. It is also important to note, that in this simple approach, a user can define the maximum value of the threshold and its lower bound. In addition, with the change of the itemset size k, a user can also control the rate of change of threshold by choosing an appropriate function of k. These functions are also very simple. For example, instead of using **vth** as,

(k / (2*(k-1))) * mt

if the function is changed to

(k / (2*(k-1)^2)) * mt

the *min_togetherness* will still be bound between **mt** and **0.5mt** but here, the rate of change of threshold will be much faster than the previous function. Hence a very simple and pragmatic method has been established for implementing variable *min_togetherness* for generating frequent itemesets.

4. CONCLUSION

Traditionally, *support* is used as the standard measure to generate the frequent itemsets for possible discovery of association rules. New and efficient algorithms have been developed for this purpose but *support* remained the default measure for frequent itemset generation. This paper has observed that the classical *support* measure considers association among items ignoring the presence of dissociation among them. As a result two itemsets of same degree and same *support* but different degree of dissociation are treated the same way in rule generation. This paper has tried to provide a pragmatic approach towards frequent itemset generation by including dissociation in the measure of association and proposed a new measure called *togetherness*. The desirable properties of *togetherness* have been established so that it can be accepted as a new and more pragmatic measure in lieu of *support*. Extending the classic Apriori algorithm, a new algorithm Aprioridis has been proposed. Performance analysis has shown that the new algorithm with *togetherness* measure really finds more desirable set of frequent itemsets than with the *support* measure when tested on the same data set.

The second part of the paper deals with the case of variable threshold. A recent work[6] has established the need of providing variable threshold instead of a fixed one for the generation of frequent itemsets. To be more precise, the threshold that accepts a candidate itemset as the frequent one should decrease as the itemset size increases. The earlier work has provided an elaborate process to generate the variable minimum *support*. Extending the principle, this paper provides a method for generating a variable minimum *togetherness*. The proposed method is not only very simple and pragmatic, but also it provides lots of control in the hand of the user. The user can provide the maximum value of the threshold, *min_togetherness*, and the lower bound of the threshold irrespective of the increase in the itemset size. The user can also control the rate of change of threshold value with the increase in itemset size by appropriately choosing a monotonically decreasing function of itemset size. Here also, performance analysis has shown how the frequent itemset generation improves from Apriori to Aprioridis (with fixed threshold) and then ultimately to Variable_Aprioridis algorithm.

The future work is supposed to take two paths. First approach is to develop computationally more efficient algorithms for frequent itemset generation with *togetherness* as the measure. Secondly, attempt will be made to extend this concept to sequential pattern mining and quantitative rule mining.

5. ACKNOWLEDGMENTS

Authors are indebted to Prof. Bimal K. Roy of Indian Statistical Institute for many useful and stimulating technical discussion on the topic of this paper.

6. REFERENCES

[1] Agrawal R., Imielinski T. and Swami A. Mining association rules between sets of items in large databases, in Proceedings of ACM SIGMOD'93 (Washington D.C, May 1993), 207-216.

[2] Agrawal R. and Srikant R. Fast Algorithms for Mining Association Rules, in Proceedings of 20th VLDB Conference (Santiago, Chile, 1994), 487-499.

[3] Cohen E., Datar M., Fujiwara S., Gionis A., Indyk P., Motwani R., Ullman J.D. and Yang C. Finding Interesting Associations without Support Pruning, in Proceedings IEEE ICDE-2000, 489-500.

[4] Pal S., Discovery of Association Rule against Dissociation, M.Tech. Dissertation Report, Indian Statistical Institute, July 2005.

[5] Park J.S., Chen M.S. and Yu P.S. An effective Hash-Based Algorithm for Mining Association Rules, in Proceedings of ACM SIGMOD'95 (San Jose 1995), 175-186.

[6] Wang K., He Y. and Han J. Pushing Support Constraints Into Association Rules Mining, IEEE Transactions on Knowledge and Data Engineering, 15, 3 (May-June 2003), 642- 657.

[7] Zaki M., Parthasarathy S. and Ogihara M. Parallel Algorithms for Discovery of Association Rules, Data Mining and Knowledge Discovery, Kluwar, 1, 4 (December 1997), 343-373.