

# Fuzzy Association Rule Mining Algorithm for Fast and Efficient Performance on Very Large Datasets

Ashish Mangalampalli\*, Vikram Pudi

Centre for Data Engineering (CDE), International Institute of Information Technology (IIIT), Hyderabad – 500 032, India.  
ashish\_m@research.iiit.ac.in, vikram@iiit.ac.in

**Abstract.** Fuzzy association rules use fuzzy logic to convert numerical attributes to fuzzy attributes, like “Income = High”, thus maintaining the integrity of information conveyed by such numerical attributes. On the other hand, crisp association rules use sharp partitioning to transform numerical attributes to binary ones like “Income = [100K and above]”, and can potentially introduce loss of information due to these sharp ranges. Fuzzy Apriori and its different variations are the only popular fuzzy association rule mining (ARM) algorithms available today. Like the crisp version of Apriori, fuzzy Apriori is a very slow and inefficient algorithm for very large datasets (in the order of millions of transactions). Hence, we have come up with a new fuzzy ARM algorithm meant for fast and efficient performance on very large datasets. As compared to fuzzy Apriori, our algorithm is *8-19 times faster* for the very large standard real-life dataset we have used for testing with various mining workloads, both typical and extreme ones. A novel combination of features like two-phased multiple-partition tidlist-style processing, byte-vector representation of tidlists, and fast compression of tidlists contribute a lot to the efficiency in performance. In addition, unlike most two-phased ARM algorithms, the second phase is totally different from the first one in the method of processing (individual itemset processing as opposed to simultaneous itemset processing at each  $k$ -level), and is also many times faster. Our algorithm also includes an effective pre-processing technique for converting a crisp dataset to a fuzzy dataset.

**Keywords:** Fuzzy association rule mining, fuzzy partitioning, fuzzy pre-processing, very large datasets, partitions, tidlists

## 1. Introduction

Crisp ARM algorithms can mine only binary attributes, and require that any numerical attributes be converted to binary attributes. Until now the general method for this conversion process is to use ranges, and try to fit the values of the numerical attribute in these ranges. Assuming we have three such ranges for the attribute Income, namely up to 20K, 20K-100K, 100K and above. Income = 50K would fit in the second partition, but so would Income = 99K. Thus, using ranges introduces uncertainty, especially at the boundaries of ranges, leading to loss of information.

A better way to solve this problem is to have attribute values represented in the interval  $[0, 1]$ , instead of having just 0 and 1, and to have transactions with a given attribute represented to a certain extent (in the range  $[0, 1]$ ). In this way crisp binary attributes are replaced by fuzzy ones [1].

Thus, we need to use fuzzy methods, by which quantitative values for numerical attributes are converted to fuzzy binary values [2], [3].

Most research in fuzzy ARM is concentrated on the theoretical aspects of finding ways to discover better fuzzy association rules, both positive as well as negative ones. Nearly all research efforts [4], [5], [6], [8], [9] in fuzzy ARM have been directed towards discovering novel interest measures, using  $t$ -norms and implicators, for fuzzy association rules, especially negative rules, and also towards finding better techniques which would facilitate mining more helpful fuzzy association rules.

Surprisingly, there has been very little effort put towards innovating fast and accurate algorithms to mine fuzzy frequent itemsets, especially from large datasets (with at least 1M transactions) and in datasets with very large number of itemsets. Until now, the only algorithms being used for fuzzy ARM are various fuzzy adaptations of Apriori [11], [12]. But, Apriori itself is a very inefficient and slow algorithm when it comes to dealing with very huge datasets [14], [15], [21]. Thus, any fuzzy adaptations of Apriori would be inadequate to deal with newer real-life datasets which are becoming larger day-by-day ( $> 1M$ ), with their fuzzy versions being even larger. Consequently, there is an acute need to innovate a fast fuzzy ARM algorithm as an alternative to fuzzy Apriori.

We have come up with a new fuzzy ARM algorithm meant for fast and efficient performance on very large datasets. Such datasets cannot be handled properly by totally in-memory algorithms like FP-Growth [13] and its fuzzy adaptations but only by algorithms which are not totally memory dependent, like ARMOR [14] and the one described in [21]. Until now, only Apriori and its various adaptations have been in use for generating fuzzy association rules from very large databases. Thus, we have compared our algorithm with fuzzy Apriori, and recorded a speedup of 8-19 times.

Generation of fuzzy association rules using an appropriate fuzzy ARM algorithm is not a straightforward process. First, we need to convert the crisp dataset, containing crisp binary and numerical attributes, into a fuzzy dataset, containing crisp binary and fuzzy binary attributes. This involves substantial pre-processing using appropriate techniques, like the one described in [20]. Conversion of numerical attributes to fuzzy binary attributes is a part of such pre-processing. Second, crisp ARM algorithms calculate the frequency of an itemset just by looking for its

presence or absence in a transaction of the dataset, but fuzzy ARM algorithms need to take into account the fuzzy membership of an itemset in a particular transaction of the dataset, in addition to its presence or absence. Thus, given a dataset, using a fuzzy ARM algorithm to generate fuzzy association rules based on this dataset is a very complex and involved process, far removed in many major as well as minor aspects from the process of generating crisp association rules, based on the same dataset, using a conventional ARM algorithm.

Our algorithm is based on a two-phased processing technique, and uses a tidlist approach for calculating the frequency of itemsets. The following are the distinctive features of our algorithm:

- Uses a different data structure, i.e. byte-vector representation of tidlist, as opposed to a list-like representation used in [14] and [21]. This adaption to the tidlist data structure has been made to accommodate fuzzy membership value along with transaction ids (tids).
- Uses compression of tidlists.
- Itemset generation and processing is done in a BFS-like fashion as in Apriori, as opposed to DFS-like in other multi-partition algorithms, like ARMOR [14].

In section 2 we briefly describe the related work, and in section 3 we describe the pre-processing methodology and fuzzy measures we have used. We put forth a detailed description of our algorithm, with an appropriate example, in section 4. Finally, before concluding, we explain our performance model setup and experimental results in sections 5.

## 2. Related Work

Most work in fuzzy ARM is directed towards theoretical aspects of finding ways to discover better fuzzy association rules, both positive as well as negative ones. [4], [5], [6] discuss in detail about fuzzy implicators and t-norms for discovering fuzzy association rules, especially negative association rules, from datasets. [6] actually talks in depth about new measures of rule quality which would help the overall fuzzy ARM process. In fact, Hüllermeier *et al.* [8], [9] make a very detailed analysis of t-norms and implicators with respect to fuzzy partitions, and provide a firm theoretical basis for their conclusions. [22] describes in great detail the general model and application of fuzzy association rules. Fuzzy Apriori, the de-facto algorithm used for fuzzy association rule mining, is used in [5], [6], [7], [10], [23]. Verlinde *et al* [7] describe in a fair amount of detail as to how fuzzy Apriori can be used to generate fuzzy association rules. Fuzzy Apriori, like Apriori, uses a record-by-record counting approach albeit the major difference being that it takes into account the fuzzy membership of an itemset in each record in order to calculate its overall count. [7] also very briefly describes a pre-processing technique, to obtain fuzzy attributes from numerical attributes, using FCM. Last, in [10] Hüllermeier

and Yi justify the relevance of fuzzy logic being applied to association rule mining in today’s data-mining setup.

## 3. Fuzzy Pre-processing and Fuzzy Measures

In this section, we describe the fuzzy pre-processing methodology and fuzzy measure that are used for the actual fuzzy ARM process. We use a data-driven pre-processing approach which automates the creation of fuzzy partitions for numerical attributes [7], and the subsequent conversion of a crisp dataset  $D$  to a fuzzy dataset  $E$ . This approach requires very minimal manual intervention even for very huge datasets.

The fuzzy sets in most real-life datasets are Gaussian-like (as opposed to triangular or trapezoidal) due to the varied and heterogeneous nature of the datasets. But, our pre-processing technique is able to generate such Gaussian-like fuzzy datasets from any real-life dataset. Numerical data present in most real-life datasets translate into Gaussian-like fuzzy sets, where in a particular data point can belong to two or more fuzzy sets simultaneously. And, this simultaneous membership of any data point in more than two fuzzy sets can affect the quality and accuracy of the fuzzy association rules generated using these data points and fuzzy sets.

In fact, most fuzzy sets of attributes found in real-life datasets are not perfectly triangular but on the contrary they are Gaussian in nature. The amount of fuzziness and Gaussian nature of fuzzy sets can be controlled using an appropriate value ( $\sim 2$ ) of the fuzziness parameter  $m$  (eq. 1).

### 3.1 Pre-processing Methodology

This pre-processing approach consists of two phases:

- Generation of fuzzy partitions for numerical attributes
- Conversion of a crisp dataset into a fuzzy dataset using a standard way of fuzzy data representation.

As part of pre-processing, we have used fuzzy c-means (FCM) clustering [16], [17], [18] in order to create fuzzy partitions from the dataset, such that every data point belongs to every cluster to a certain degree  $\mu$  in the range [0, 1]. The algorithm tries to minimize the objective function:

$$\sum_{i=1}^N \sum_{j=1}^C \mu_{ij}^m \|x_i - c_j\|^2 \quad (1)$$

where  $m$  is any real number such that  $1 \leq m < \infty$ ,  $\mu_{ij}$  is the degree of membership of  $x_i$  in the cluster  $j$ ,  $x_i$  is the  $i^{th}$   $d$ -dimensional measured data,  $c_j$  is the  $d$ -dimension center of the cluster, and  $\|*\|$  is any norm expressing the similarity between any measured data and the center. The fuzziness parameter  $m$  is an arbitrary real number ( $m > 1$ ).

In the first phase, we apply one-dimensional FCM clustering on each of the numeric attributes to obtain the corresponding fuzzy partitions, with each value of any

numeric attribute being uniquely identified by its membership function  $\mu$  in these fuzzy partitions. One needs to select appropriate value of  $k$  (number of one-dimensional clusters), and then label the resulting clusters according to the nature of the attribute.

In the second phase, if an attribute is quantitative, then the pre-processing methodology converts each crisp record in the dataset  $D$  to multiple fuzzy records based on the number of fuzzy partitions defined for the attribute. Doing so has the potential of generating fuzzy records in a combinatorial explosive manner. To deal with this problem, we have fixed a lower threshold for the membership function  $\mu$  ( $\approx 0.1$  most of the times) to keep the number of fuzzy records generated under control. If an attribute is a binary attribute, then we output each record appended with a membership function  $\mu = 1$ , indicating this value has full membership in the binary attribute. The final fuzzy dataset  $E$  is used as input to our algorithm.

### 3.2 Fuzzy Association Rules and Fuzzy Measures

During the fuzzy ARM process, a number of fuzzy partitions are defined on the domain of each quantitative attribute, as result of which the original dataset is transformed into an extended one with attribute values in the interval  $[0, 1]$ . In order to process this extended (fuzzy) dataset, we need new measures (analogous to crisp support and confidence), which are in terms of t-norms. The generation of fuzzy association rules is directly impacted by the fuzzy measures we use.

Equations 2 and 3 respectively define a t-norm and the cardinality of a fuzzy set in a finite universe  $D$  [4], [5], [7]. Fuzzy sets  $A$  and  $B$  in  $D$  are mapped as  $D \rightarrow [0, 1]$ , with  $A(x)$  and  $B(x)$  being the degrees to which attributes  $A$  and  $B$  are present in a transaction  $x$  respectively. Thus, using fuzzy partitions  $A$  and  $B$  and a t-norm, we can define fuzzy support (eq. 4) and confidence (eq. 5) [4], [5], [7]. The more generally used t-norms are listed in Table I, with  $T_M$  t-norm being the most popular one. Even we use the same t-norm in our algorithm.

$$A \cap_T B(x) = T(A(x), B(x)) \quad (2)$$

$$|A| = \sum_{(x \in D)} A(x) \quad (3)$$

$$\text{supp}(A \Rightarrow B) = \sum_{(x \in D)} (A \cap_T B)(x) / |X| \quad (4)$$

$$\text{conf}(A \Rightarrow B) = \sum_{(x \in D)} (A \cap_T B)(x) / \sum_{(x \in D)} A(x) \quad (5)$$

**Table I. t-norms in fuzzy sets**

t-norm
$T_M(x, y) = \min(x, y)$
$T_P(x, y) = xy$
$T_W(x, y) = \max(x + y - 1, 0)$

## 4. Fuzzy Association Rule Generation for Very Large Datasets

Our algorithm uses two phases in a partition-approach to generate fuzzy association rules. The dataset is logically divided into  $p$  disjoint horizontal partitions  $P_1, P_2, \dots, P_p$ . Each partition is as large as can fit in available main memory. For ease of exposition, we assume that the partitions are equal-sized, though each partition could be of any arbitrary size as well.

We use the following notations:

- $E$  = fuzzy dataset generated after pre-processing
- Set of partitions  $P = \{P_1, P_2, \dots, P_p\}$
- $tid[it]$  = tidlist of itemset  $it$
- $\mu$  = fuzzy membership of any itemset
- $count[it]$  = cumulative  $\mu$  of itemset  $it$  over all partitions in which  $it$  has been processed
- $d$  = number of partitions (for any particular itemset  $it$ ) that have been processed since the partition in which  $it$  was added (including the current partition and the partition in which  $it$  was added).

0.12	0.23	0.00	0.00	0.50	ABC
0.90	0.30	0.00	0.11	0.00	BCD
0.12	0.23	0.00	0.00	0.00	ABCD

**Fig. 1. Byte-vector-like data structure for tidlists**

The byte-vector-like data structure we use is illustrated in fig. 1. Each cell of the byte-vector stores  $\mu$  of the itemset corresponding to the cell index of the tid to which the  $\mu$  pertains. Thus, the  $i^{th}$  cell of the byte-vector contains the  $\mu$  for the  $i^{th}$  tid. If a particular transaction does not contain the itemset under consideration, the cell corresponding to that transaction is assigned a value of 0. When the byte-vector is initialized, each cell by default has 0.

We convert  $\mu$ , a floating-point number, to a byte by multiplying it by 100, and taking the ceiling of the result to get the final byte representation of the  $\mu$ , which is actually stored in the byte-vector. Thus, each  $\mu$  can be represented with a precision of two digits after the decimal point. Whenever we need the actual floating-point value of  $\mu$ , we just divide the value obtained from the byte-vector by 100. In this way, we achieve a lot of saving in main memory space, and thus speed up the execution of the algorithm.

Tidlists, especially in byte-vector representation, can be very huge, and may cause all main memory to be completely used up leading to incessant thrashing. We have overcome this problem by using an appropriate compression algorithm (*zlib* compression algorithm) which provides very good inflation and deflation speeds. Though other ARM algorithms, like VIPER [19], have used various compression techniques, the compression process has not been efficient and fast in all possible scenarios of operation.

But, using our compression technique, we have achieved considerable speed up for sorts of tidlists, and are able to get any dataset processed with far fewer number of partitions than would have been required in case no compression were used.

#### 4.1 First Phase

In the first phase, F-ARMOR scans each transaction in the current partition of the dataset, and constructs a tidlist for each singleton found. After all singletons in the current partition have been enumerated, the tidlists of singletons which are not  $d$ -frequent are dropped (fig. 2, lines 1-9). *An itemset is  $d$ -frequent if its frequency over  $d$  partitions equals or exceeds the support adjusted for  $d$  partitions, i.e. it is frequent over  $d$  partitions of the dataset (where  $d$  ≤ number of partitions in the dataset).*

The count of each singleton  $s$  is maintained in  $count[s]$ . To generate larger itemsets, we use breadth-first search (BFS) technique in a fashion similar to one used in Apriori (fig. 2, lines 10-18). At the  $k^{\text{th}}$  level, each  $k$ -itemset  $it_k$  is combined with another  $k$ -itemset  $it_k'$  to generate a  $(k+1)$ -itemset  $it_{k+1}$ , if the two  $k$ -itemsets differ by just one singleton. The tidlist  $td[it_{k+1}]$  for each  $(k+1)$ -itemset  $it_{k+1}$  is generated by intersecting the tidlists of its parent  $k$ -itemsets,  $td[it_k]$  and  $td[it_k']$ . If  $it_{k+1}$  is not  $d$ -frequent, then  $td[it_{k+1}]$  is discarded. Additionally, the count of each  $(k+1)$ -itemset  $it_{k+1}$  is maintained in  $count[it_{k+1}]$ .

During the intersection of the parent tidlists, for each cell with index  $l$  of the two parent tidlists, we obtain the minimum of the two fuzzy membership values the  $l^{\text{th}}$  cells of the parent tidlists to form the resultant fuzzy membership of  $l^{\text{th}}$  cell of the child tidlist. By taking the minimum, we are in effect applying the  $T_M$  fuzzy t-norm. Any other t-norm could also have been used. Similarly, the intersection is done for each cell of the parent tidlists to obtain the child tidlist.

Tidlist creation is done as soon as an itemset has been created. After  $it_k$  has been combined with all other  $k$ -itemsets, its tidlist can be discarded. By doing so, we save main memory space, and reduce the number of inflations (all tidlists reside in memory in compressed state). Thus, for each level  $k$ , all  $(k+1)$ -itemsets and corresponding tidlists are generated, until at any level  $k$ , we have only one or no  $d$ -frequent itemset. Then we traverse the next partition and process it in a similar manner, till all partitions have been processed, signaling the end of the first phase.

#### 4.2 Second Phase

Then we move on to the second phase of the algorithm. The second phase is quite different from the first one in many aspects. First, we traverse each partition one-by-one starting from the first partition. All itemsets added in the current partition in the first phase have been enumerated over the whole dataset  $E$ , and thus can be removed. Of these removed itemsets, ones which are frequent over the whole dataset  $E$  are output (fig. 3, lines 1-5).

Then, for each remaining itemset  $it$ , we identify its constituent singletons  $s_1, s_2, \dots, s_m$  and then obtain the tidlist of  $it$   $td[it]$  by intersecting the tidlists of all the constituent singletons. Additionally, the count of each singleton  $it$  is updated in  $count[it]$  (fig. 3, lines 6-10). Thus, we alternate between outputting and deleting itemsets, and creating tidlists for itemsets, until no more itemsets are left. Then the algorithm terminates (fig. 3, lines 11-12).

```

1) traverse each partition  $P$ 
2)   traverse each transaction  $t$  in current partition  $P$ 
3)     for each singleton  $s$  in
         current transaction  $t$ 
4)       calculate  $\mu$  for each  $s$ 
5)        $count[s] += \mu$ 
6)       add  $t$  & corresponding  $\mu$  for  $s$  to tidlist  $td[s]$ 
7)   for each singleton  $s$ 
8)     if  $s$  is not  $d$ -frequent
9)       delete  $td[s]$ 
10)  while no. of  $d$ -freq  $k$ -itemsets at each  $k$ -level  $\geq 2$ 
11)    for each possible pair of itemsets  $it_k$  and  $it_k'$ 
12)      if  $it_k$  and  $it_k'$  differ exactly by 1 singleton
13)        combine  $it_k$  and  $it_k'$  to get  $it_{k+1}$ 
14)         $td[it_{k+1}] = td[it_k] \cap td[it_k']$ 
         /* use t-norm  $T_M$  for intersection */
15)        calculate  $\mu$  for  $it_{k+1}$  using  $td[it_{k+1}]$ 
16)         $count[it_{k+1}] += \mu$ 
17)        if  $it_{k+1}$  is not  $d$ -frequent
18)          delete  $td[it_{k+1}]$ 

```

Fig. 2. Pseudo-code for Phase 1

```

1) traverse each partition  $P$ 
2)   for each itemset  $it$  to  $P$  in 1st phase
3)     if  $it$  is frequent (based on  $count[it]$ )
         over the whole dataset  $E$ 
4)       output( $it$ )
5)       remove  $it$ 
6)   for each remaining itemset  $it$ 
7)     identify constituent singletons
          $s_1, s_2, \dots, s_m$  of  $it \forall it = s_1 \cap s_2 \cap \dots \cap s_m$ 
8)     tidlist  $td[it] =$  intersect tidlists of
         all constituent singletons
         /* tidlist intersection is same as in phase 1 */
9)     calculate  $\mu$  for  $it$  using  $td[it]$ 
10)     $count[it] += \mu$ 
11)  if no itemsets remain to be enumerated
12)    exit

```

Fig. 3. Pseudo-code for Phase 2

#### 4.3 Illustration of F-ARMOR and Fuzzy Association Rule Generation

Let us take an example of two itemsets  $p$  (“ABC”) and  $q$  (“BCD”). Their child itemset would be  $r$  (“ABCD”). Byte-vector-style representation of their tidlists is illustrated in fig. 1. Assuming that there are five partitions, and that  $r$

became  $d$ -frequent and was added in the third partition in the first phase. Tidlists for  $r$  are generated in the third through fifth partitions, assuming that no parent of  $r$  goes into the negative border. Thus, at the end of the first phase, the frequency of  $r$  is available for the third through fifth partitions.

In the second phase, we enumerate the frequency of  $r$  in partitions 1 and 2. We generate tidlists of all singletons possible for each of these two partitions. And, then we generate tidlists for  $r$  once in each of the two partitions 1 and 2, by taking an intersection of the tidlists of the singletons involved in  $r$ , *i.e.* “A”, “B”, “C”, and “D”. Thus, at the start of processing of the third partition (in the second phase),  $r$  would have been enumerated over the whole dataset  $E$ , and would be deleted. It would also be output, if it is frequent over the whole dataset  $E$ . All other itemsets are processed in a similar manner. The algorithm terminates, when there are no more itemsets left to be processed at start of a partition (in the second phase).

## 5. Performance Study

In this section, we assess the performance of our algorithm with respect to fuzzy Apriori, which is the most popular and widely used online fuzzy mining algorithm. We have used the USCensus1990raw dataset for our experimental analysis. The crisp dataset has around 2.5M transactions, and the fuzzy dataset has around 10M transactions. Thus, the dataset size is significantly larger than the available main memory. We have used 12 attributes present in the dataset, of which eight are quantitative and the rest are binary. For each of the eight quantitative attributes, we have generated fuzzy partitions using FCM, and then generated the fuzzy version of the dataset (using a threshold for membership function  $\mu$  as 0.1) as detailed in Section 3.

Our experiments cover various mining workloads, both typical and extreme ones, with various values of support. The performance metric in all the experiments is the total execution time taken by the mining operation. We have performed each of the experiments on two different computers with different configurations, especially in terms of RAM sizes:

- Computer  $C_1$  – AMD Sempron 2600+ (1.6 GHz), 512 MB DDR RAM, and PATA 7200 RPM HDD.
- Computer  $C_2$  – Intel Core 2 Duo 2.8 GHz, 1 GB DDR2 RAM, SATA 7200 RPM HDD.

### 5.1. Experimental Results

Fig. 4 illustrates the results obtained on  $C_1$  on the USCensus1990raw dataset, using various values of support ranging from 0.075 to 0.4. It can be clearly observed that F-ARMOR performs *8-19 times faster* than fuzzy Apriori, depending on the support used. Please note that the execution times for fuzzy Apriori for support values 0.075 and 0.1 have not been calculated as the time exceeded 50K seconds. Fig. 5 illustrates the results obtained by running the same experiment on  $C_2$ . On this computer, we observe

that our algorithm has speeds nearly *8-13 times* that of fuzzy Apriori.

More importantly, for any dataset there is a particular support value for which optimal number of itemsets is generated and for supports less than this value, we get a flood of itemsets which are of no practical use. From our experiments, we have observed that our algorithm performs most *efficiently* and *speedily* at this *optimal support value*, which occurs in the range of 0.15 - 0.2 for this dataset.

Our endeavor was to reduce the number of partitions as much as possible, and have actually executed our algorithm with *just one partition* for support values 0.2 – 0.4 on  $C_1$ , and 0.1 – 0.4 on  $C_2$ , without the need of the second phase. Less number of partitions means faster processing and less consumption of resources like main memory and processor. If we try to forcibly increase the number of partitions for a dataset (with a specific support), which can be processed with lesser number of partitions, we end up increasing the overall time required for processing. In such a case, more number of partitions means that there is a greater chance of more itemsets becoming  $d$ -frequent (though finally only overall frequent itemsets are output), and more time is spent generating and analyzing far greater number of itemsets.

For other support values, the number of partitions is *close to 1 as possible*, keeping in mind that the main memory is utilized in the best manner possible, without any thrashing. Moreover, the time taken by the second phase was found to be negligible as compared to that taken by the first phase. From these experiments, the time taken by the second phase ranges from *0.02 – 0.1 times* (depending on the support value and number of partitions used) that taken by the first phase. Further more, with our compression technique, we have got compressed tidlists which are *2% - 20%* the sizes of uncompressed tidlists.

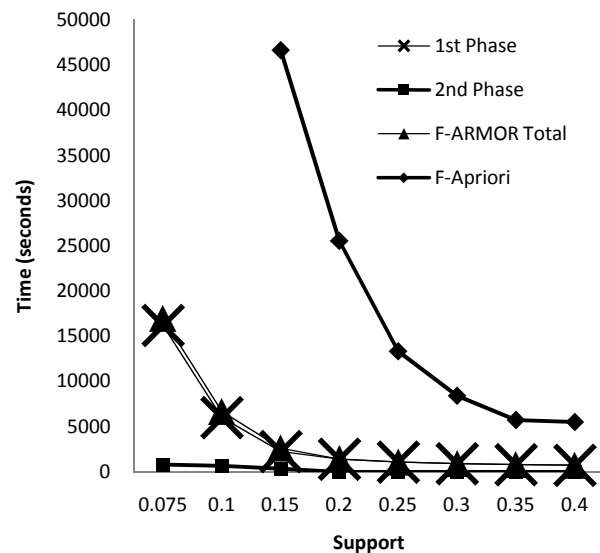


Fig. 4. Experiment Results on  $C_1$

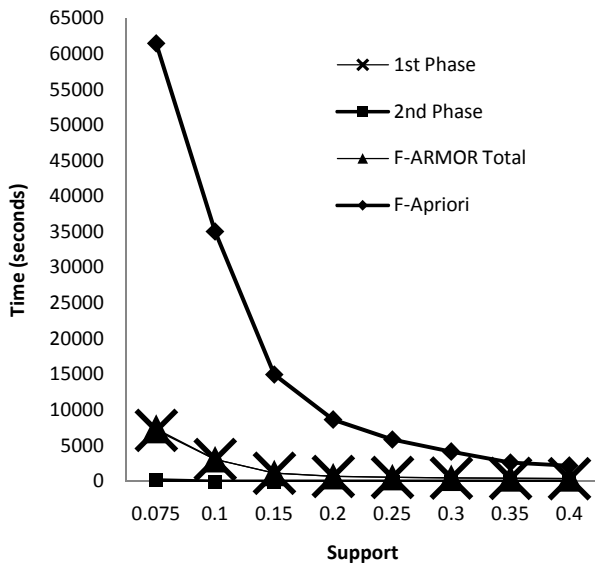


Fig. 5. Experiment Results on C<sub>2</sub>

## 6. Conclusions

We have presented a novel fuzzy ARM algorithm, for very huge datasets, as an alternative to fuzzy Apriori, which is the most widely used algorithm for fuzzy ARM. Through our experiments, we have shown that our algorithm is 8-19 times faster than fuzzy Apriori. This considerable speed up has been achieved because novel properties like two-phased tidlist-style processing using partitions, tidlists represented in the form of byte-vectors, effective compression of tidlists, and a tauter and quicker second phase of processing.

## References

- Zadeh, L. A.: Fuzzy sets. *Inf. Control*, 8, 338-358 (1965).
- Chen G., Yan P., Kerre E.E.: Computationally Efficient Mining for Fuzzy Implication-Based Association Rules in Quantitative Databases. *International Journal of General Systems*, 33, 163-182 (2004).
- Hüllermeier, E.: Fuzzy methods in machine learning and data mining: Status and prospects. *Fuzzy Sets and Systems*. 156, 387-406 (2005).
- De Cock, M., Cornelis, C., Kerre, E.E.: Fuzzy Association Rules: A Two-Sided Approach. In: *FIP*, pp 385-390 (2003).
- Yan, P., Chen, G., Cornelis, C., De Cock, M., Kerre, E.E.: Mining Positive and Negative Fuzzy Association Rules. In: *KES*, pp. 270-276. Springer (2004).
- De Cock, M., Cornelis, C., Kerre, E.E.: Elicitation of fuzzy association rules from positive and negative examples. *Fuzzy Sets and Systems*, 149, 73-85 (2005).
- Verlinde, H., De Cock, M., Boute, R.: Fuzzy Versus Quantitative Association Rules: A Fair Data-Driven Comparison. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 36, 679-683 (2006).
- Dubois, D., Hüllermeier, E., Prade, H.: A systematic approach to the assessment of fuzzy association rules. *Data Min. Knowl. Discov.*, 13, 167-192 (2006).
- Dubois, D., Hüllermeier, E., Prade, H.: A Note on Quality Measures for Fuzzy Association Rules. In: *IFSA*, pp. 346-353. Springer-Verlag (2003).
- Hüllermeier, E., Yi, Y.: In Defense of Fuzzy Association Analysis. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 37, 1039-1043 (2007).
- Agrawal, R., Imielinski, T., Swami, A.N.: Mining Association Rules between Sets of Items in Large Databases. *SIGMOD Record*, 22, 207-216 (1993).
- Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules. In: *VLDB*, pp. 487-99. Morgan Kaufmann (1994).
- Han, J., Pei, J., Yin, Y.: Mining Frequent Patterns without Candidate Generation. In: *SIGMOD Conference*, pp 1-12. ACM Press (2000).
- Pudi, V., Haritsa, J.: ARMOR: Association Rule Mining based on Oracle. *CEUR Workshop Proceedings*, 90 (2003).
- Pudi, V., Haritsa, J.: On the optimality of association-rule mining algorithms. Technical Report TR-2001-01, DSL, Indian Institute of Science (2001).
- Dunn, J. C.: A Fuzzy Relative of the ISODATA Process and its Use in Detecting Compact, Well Separated Clusters. *J. Cyber.*, 3, 32-57 (1974).
- Hoppner, F., Klawonn, F., Kruse, R., Runkler, T.: *Fuzzy Cluster Analysis, Methods for Classification, Data Analysis and Image Recognition*. Wiley, New York (1999).
- Bezdek, J. C.: *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers, Norwell, MA (1981).
- Shenoy, P., Haritsa, J., Sudarshan, S., Bhalotia, G., Bawa, M., Shah, D.: Turbo-charging Vertical Mining of Large Databases. In: *SIGMOD Conference*, pp 22-33. ACM Press (2000).
- Mangalampalli, A., Pudi, V.: Fuzzy Logic-based Pre-processing for Fuzzy Association Rule Mining. Technical Report IIIT/TR/2008/127, International Institute of Information Technology (2008).
- Savasere, A., Omiecinski, E., Navathe, S.B.: An Efficient Algorithm for Mining Association Rules in Large Databases. In: *VLDB*, pp. 432-444. Morgan Kaufmann (1995).
- Delgado, M., Marin, N., Sanchez, D., Vila, M. A.: Fuzzy Association Rules: General Model and Applications. *IEEE Transactions on Fuzzy Systems* 11, 214-225 (2003).
- Shu-Yue, J., Tsang, E., Yenng, D., Daming, S.: Mining fuzzy association rules with weighted items. In: *IEEE International Conference on SMC*, pp. 1906-1911, IEEE (2000).