ABSTRACT
The mining of frequent itemsets is the prerequisite and most time consuming process for association rule mining. At now, most efficient Apriori-like algorithms rely heavily on the minimum support constraints to prune the vast amount of non-candidate itemsets. These algorithms store many unwanted itemsets and transactions. In this paper proposes a novel frequency itemsets generation algorithm called TRApriori that maintains its performance even at relative low supports. Here transaction reduction technique helps to perform better. Experimental evaluations also show that our TRApriori algorithm on an outset is faster than Apriori-like algorithms and related algorithm.

KEYWORDS
Frequent itemsets, Minimum support, Apriori, AprioriTID, HEA, TRApriori.

1. INTRODUCTION
Data Mining refers to extracting or mining knowledge from large amounts of data. In 1993, Agarwal, Imieliński, and Swami introduced a class of regularities, association rules and gave an algorithm for finding such rules [3]. The typical example for association rule mining is Market Basket analysis. The Bar code technology has made it possible for retail organizations to collect and store massive amounts of sales data. A record in it typically consists of itemsets bought in each transaction in each row. Finding association rules for such basket data will improve the sales and cross marketing. Other applications include store layout, add on sales and customer segmentation.

In general, the work of association rule can be decomposed into two phases:
(1) Frequent itemsets generation - find out all itemsets that sufficiently exceed the minsup, and (2) Rules construction - from the frequent itemsets, generate all association rules having confidence higher than the minconf. Since the second phase is straightforward and less expensive. This paper concentrates only on the first phase for finding all frequent itemsets.

In this paper have proposed a new algorithm called TRApriori to efficiently mine frequent itemsets. This paper have also proved that our algorithm TRApriori has an edge over existing algorithms like Apriori, AprioriTID, HEA (High Efficient AprioriTid) and FP-Growth in terms of transaction reduction and speed. The rest of the paper is organized as follows. A review of the previous works regarding AprioriTid and HEA are given in the next section. Then This paper will describe the proposed theorem with proof and proposed algorithm called TRApriori for finding frequent item sets. Empirical evaluations and results of our algorithm (TRApriori) over other algorithms using FIMI’s dataset (retail.dat) and other synthetic data sets are described in the next section. Conclusions are stated in the last section.

2. PREVIOUS WORKS
2.1. AprioriTID Algorithm
The AprioriTID [2] is another variant of Apriori which reduces the time needed for the frequency counting procedure by replacing every transaction in the database by the set of candidate itemsets that occurs in that transaction. This is done repeatedly at every iteration k. The adapted transaction database is denoted by $\overline{C_k}$. Although the AprioriTID algorithm is much faster in later iterations, it performs much slower than Apriori in early iterations. This is mainly due to the additional overhead that is created when $\overline{C_k}$ does not fit into main memory and has to be written to disk [7-10]. If a transaction does not contain any candidate k-sets, then $\overline{C_k}$ will not have an entry for this transaction. Hence, the number of entries in $\overline{C_k}$ may be smaller than the number of transactions in the database, especially at later iterations of the algorithm. The AprioriTid algorithm also uses the Apriori-gen function to determine the candidate itemsets before the pass begins.
The transaction in \( \overline{C_k} \) will be of the form \(<TID, \{X_k\}>\), where each \( X_k \) is a potentially a large \( k \)-itemset present in the transaction with identifier TID. The member of \( \overline{C_k} \) corresponding to the transaction \( t \) is defined as

\[
\overline{C_k} = < t.TID, \{ c \in C_k | c \ contained \ contained \} >
\]

The drawbacks of AprioriTID are that the modified data structure can be much larger than the initial database and only faster in the later stage of the passes.

2.2. HEA Algorithm

**HEA** [1] is the improved algorithm of AprioriTID. Here, the member of \( \overline{C_k} \) corresponding to the transaction \( t \) is defined as

\[
\text{\overline{C_k} = < t.TID, \{ c \in C_k | c \ contained \ contained \} >.}
\]

This considerably decreases the size of stored data in \( \overline{C_k} \). Moreover, searching data scale is reduced when computing support of itemsets in \( C_k \). At the same time, it reduces much time of I/O and running. The process is described as follows:

(a) Confirming itemset \( c \) in \( C_k \), then the transaction set \( T_c \) presented with TID including the item of \( c \), is computed. (b). The number of entries computed in \( T_c \), defined as the \( |T_c| \), which is support of itemset \( c \). (c) If \( |T_c| \geq \text{minsup} \), \( c \) is included into \( L_k \) and \( \overline{C_k} \), otherwise deleting \( c \).

Through the above process, with the computed support in \( C_k \), void itemset can be directly deleted from \( \overline{C_k} \) or added to \( \overline{C_k} \) and \( L_k \). After analyzing this algorithm, we found that HEA can be further improved by reducing unwanted transactions and itemsets which is explained in next section through proposed new algorithm TRApriori.

3. TRApriori Algorithm

TRApriori - Improved Transaction Reduction algorithm in Figure 1 is an improvement algorithm of AprioriTID and HEA whose idea is based on the theorem described in next sub section. Here relationship of transaction \( t \) with the entry in \( \overline{C_k} \) is defined as

\[
\overline{C_k} = < t.TID, \{ c \in C_k | c \ contained \ contained \} >
\]

Each member of the set \( \overline{C_k} \) will be of the form \(<TID, \{X_k\}>\) where each \( X_k \) is a potentially large \( k \)-itemset present in the transaction with identifier TID.

3.1. Theorem and Proof

**Theorem:** If \( c \in C_{k-1} \) and \( c \).support < \( \text{minsup} \), \( T_{\text{items}} \leq k \), \( m = 1 \), then \( c \) is useless in \( \overline{C_k} \) where \( \overline{T_{\text{items}}} \) is total itemcount in each transaction and \( m \) is the no. of combinations.

**Proof:** 1. [For \( T_{\text{items}} \leq k \)]

From the Apriori-gen algorithm, it is known that \( C_k \) is formed from \( L_{k-1} \). Consider a transaction \( T_i = \{ a_1, a_2, \ldots, a_m \} \). The candidate generation would lineup from

\[
C_1 = \{ a_1 \}, \{ a_2 \}, \ldots, \{ a_m \};
C_2 = \{ a_1, a_2 \}, \{ a_1, a_3 \}, \ldots, \{ a_1, a_m \}, \{ a_2, a_3 \}, \ldots, \{ a_m \};
C_3 = \{ a_1, a_2, a_3 \}, \{ a_1, a_2, a_4 \}, \ldots, \{ a_2, a_3, a_4 \}, \ldots, \{ a_m-2, a_m \};
C_m = \{ a_1, a_2, a_3, \ldots, a_m \}
\]

and it will go up to \( C_{m+1} \). So \( \{ a_1, a_2, a_3, \ldots, a_m \} \) will not appear in \( C_{m+1} \). That is to say that we need not compute the \( C_{m+1} \) in \( T_i \).

Hence, \( \forall T_i \in \overline{C_k+1}, T_i \) need not be taken into account. So, \( T_i \) can be rejected in \( \overline{C_{k+1}} \). Hence the proof.

2. [For \( m = 1 \), where \( m \) is no. of combinations]

Now consider the same transaction \( T_i = \{ a_1, a_2, a_3 \ldots a_m \} \).
In general the candidate generation would have

\[
\{ a_1 \}, \{ a_2 \}, \ldots, \{ a_1, a_2 \}, \ldots, \{ a_1, a_m \}, \ldots, \{ a_2, a_3 \}, \ldots, \{ a_m-2, a_m \}
\]

If all the itemsets in \( C_k \) except one is rejected during \( L_{m-1} \) due to above hypothesis (say \( A_b \) remains) then \( \overline{C_{k+1}} \) will contain only \( \{A_b\} \). So \( \{A_b\} \) alone will not able to form a candidate in \( \overline{C_{k+1}} \). Hence in such \( \forall T_i \in \overline{C_{k+1}}, T_i \) can be rejected in \( \overline{C_{k+1}} \). Hence the proof.

Figure 1. TRApriori Algorithm

**INPUT:** \( D, \text{minsup} \)

**OUTPUT:** \( L (D, \text{minsup}) \)

1) \( L_1 = \{ \text{large l-itemsets} \}; \)
2) \( \overline{C_1} = \text{Database} \ D; \) (With all items not in \( L_1 \) and \( \forall t \in T_\text{items} = 1 \) removed)
3) For (k=2; \( \overline{C_{k-1}} \neq \emptyset \); k++) do begin
4) \( C_k = \text{Apriori-gen} (L_{k-1}) \) // New candidates
5) \( \overline{C_k} = \emptyset ; \)
6) For all \( c \in C_k \) do begin
7) \( \overline{C} = \emptyset ; \)
8) \( T_c = \{ t.TID \mid t \in \overline{C_{k-1}}, (c - c[k]) \in T_\text{set-of-itemsets} \wedge (c - c[k]) \in T_\text{set-of-itemsets} \} \)
9) If \( |T_c| \geq \text{minsup} \) then begin
A Better Performed Transaction Reduction Algorithm For Mining Frequent Itemsets From Large Voluminous Database

10) \[ L_k = \bigcup_{\{c\}} \{c\}; \]
11) For all \( p \in T_c \) do
12) \[ \text{If } (T_{\text{items}} > k) \text{ then begin} \]
13) \[ \bigcup C = k < p, c >; \]
14) \[ \text{end} \]
15) \[ \text{end} \]
16) \[ \text{If } |C| \neq 1 \text{ then begin} \]
17) \[ \bigcup C = C; \]
18) \[ \text{end} \]
19) \[ \text{end} \]
20) \[ \text{end} \]
21) \[ \text{end} \]
22) Ans = \[ \bigcup C \]

3.2. TRApriori Algorithm Details

Let us consider the following example in Table 1 to show how TRApriori works. The algorithm starts by scanning the database to generate 1- frequent itemset. \( L_1 \). \( \bigcup C \) is generated with itemsets that are present in \( L_1 \). For \( k = 1 \), \( \bigcup C \) corresponds to the database \( D \) although conceptually each item \( I \) is replaced by the itemset \( \{i\} \).

For \( k > 1 \), \( C_K \) is generated by the apriori-gen algorithm. Confirming itemset \( c \) in \( C_k \), then the transaction set \( T_c \) presented with TID, including itemset \( c \) is computed. The number of entries is computed in \( T_c \) defined as \( |T_c| \) which is support of itemset \( c \). If \( |T_c| \geq \text{minsup} \), \( c \) is included into \( L_k \) and \( \bigcup C \), otherwise deleting \( c \).

### Table 2. L1

<table>
<thead>
<tr>
<th>Item set</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

### Table 3. \( \bigcup C_1 \)

<table>
<thead>
<tr>
<th>Tid</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>T100</td>
<td>{5}, {6}, {8}</td>
</tr>
<tr>
<td>T200</td>
<td>{2}, {4}, {8}</td>
</tr>
<tr>
<td>T300</td>
<td>{4}, {5}, {7}</td>
</tr>
<tr>
<td>T400</td>
<td>{2}, {3}</td>
</tr>
<tr>
<td>T500</td>
<td>{5}, {6}, {7}</td>
</tr>
<tr>
<td>T600</td>
<td>{2}, {3}, {4}</td>
</tr>
<tr>
<td>T700</td>
<td>{2}, {6}, {7}</td>
</tr>
<tr>
<td>T1000</td>
<td>{3}, {5}, {7}</td>
</tr>
<tr>
<td>T1100</td>
<td>{3}, {5}, {7}</td>
</tr>
<tr>
<td>T1200</td>
<td>{5}, {6}, {8}</td>
</tr>
<tr>
<td>T1300</td>
<td>{2}, {4}, {6}, {7}</td>
</tr>
<tr>
<td>T1400</td>
<td>{3}, {5}, {7}</td>
</tr>
<tr>
<td>T1500</td>
<td>{2}, {3}</td>
</tr>
</tbody>
</table>

If the total item count of a transaction \( T \) items \( \leq k \), then that transaction \( t \) is deleted from \( \bigcup C_{k-1} \). Also, if \( |\bigcup C_k| = 1 \), then it is also not included in \( \bigcup C_k \) and that transaction is deleted from \( \bigcup C_{k-1} \). Thus the entries in \( \bigcup C_k \) may be smaller than the transaction in database especially for large values of \( k \). The corresponding \( L_K \) and \( \bigcup C \) are shown in Tables 2, 3, 4, 5, 6 and 7.

### Table 4. L2

<table>
<thead>
<tr>
<th>Item set</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2,3}</td>
<td>3</td>
</tr>
<tr>
<td>{2,4}</td>
<td>3</td>
</tr>
<tr>
<td>{3,5}</td>
<td>3</td>
</tr>
<tr>
<td>{3,7}</td>
<td>3</td>
</tr>
<tr>
<td>{5,6}</td>
<td>3</td>
</tr>
<tr>
<td>{5,7}</td>
<td>3</td>
</tr>
<tr>
<td>{6,7}</td>
<td>3</td>
</tr>
</tbody>
</table>

### Table 5. \( \bigcup C_2 \)
Table 6. L3

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>T500</td>
<td>{5,7}, {5,6}, {6,7}</td>
</tr>
<tr>
<td>T600</td>
<td>{2,3}, {2,4}</td>
</tr>
<tr>
<td>T1000</td>
<td>{3,5}, {3,7}, {5,7}</td>
</tr>
<tr>
<td>T1100</td>
<td>{3,5}, {3,7}, {5,7}</td>
</tr>
<tr>
<td>T1300</td>
<td>{2,4}, {6,7}</td>
</tr>
<tr>
<td>T1400</td>
<td>{3,5}, {3,7}, {5,7}</td>
</tr>
</tbody>
</table>

Table 7. \( \overline{C}_3 \)

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

4. EXPERIMENTAL RESULTS

In order to evaluate the performance of TRApriori, we have tested various datasets like FIMI data set [16] and synthetic dataset and compared them with Apriori, and FP growth whose implementations were used from [17] and [18]. Since no public implementation of AprioriTID and HEA is available we used our own implementations to compare with TRApriori algorithm.

4.1. Using Synthetic Dataset

Using a sample Database of 1020 transactions with 9 items we tested TRApriori with AprioriTID and HEA. With the minimum support as 20%, the results are as shown in the Table 8 [11 – 15] and figure 2.

Table 8. Contrast of Record numbers

<table>
<thead>
<tr>
<th>Records</th>
<th>AprioriTID</th>
<th>HEA</th>
<th>TRApriori</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \overline{C}_1 )</td>
<td>2720</td>
<td>2720</td>
<td>2584</td>
</tr>
<tr>
<td>( \overline{C}_2 )</td>
<td>2584</td>
<td>1524</td>
<td>1088</td>
</tr>
<tr>
<td>( \overline{C}_3 )</td>
<td>340</td>
<td>204</td>
<td>0</td>
</tr>
<tr>
<td>( \overline{C}_4 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

From the Table 8 it is observed that number of records is averagely reduced to 35 % when compared to AprioriTID and 18 % when compared to HEA. Hence the execution will also be considerably reduced. The graph in the Figure 2 is plotted with the help of the Table 8.

This paper also compared our TRApriori algorithm with Apriori using the synthetic dataset whose transaction varies from 1000 to 500000 and with a constant minimum support of 3% in Figure 3.

4.2. Using FIMI’s retail.dat Dataset

This paper used the dataset retail.dat from FIMI site [16] with 17822 transactions and 9999 items. The experiment is performed on a workstation with processor clock cycle 2.0 GHz, 512 MB of main memory. The Figure 4 shows the result up to the minimum support of 8%.
A Better Performed Transaction Reduction Algorithm For Mining Frequent Itemsets From Large Voluminous Database

5. CONCLUSION
This paper proposed a novel data structure, a TRApriori algorithm to mine frequent itemsets. The advantages of TRApriori over existing algorithms include (1) Interactive mining with different supports (2) Faster execution time (3) Infrequently used item are not stored and hence improves the size of the query data. We have implemented TRApriori algorithm and compared our approach with other known algorithms. From the experimental results, TRApriori is shown to be more efficient and scalable to large amount of transactions and out perform other algorithms.

FUTURE SCOPE
Extension of this new approach will be applied to other algorithms like Eclat for further efficiency.

REFERENCES
[13] Survey on Frequent Pattern Mining, Bart Goethals, HIIT Basic Research Unit, University of Helsinki, Finland.
[15] Jiawei Han, Jian Pei and Yiwen Yin, Mining Frequent Pattern without Candidate Generation, School of Computing Science, Simon Fraser University.
[19] Data Mining – Concepts and Techniques, Jiawei Han, Micheline Kamber.
Authors

E. Ramaraj is presently working as a Director, computer centre at Alagappa university, Karaikudi. He has 20 years teaching experience and 5 years research experience. He has presented research papers in more than 20 national and international conferences and published more than 30 papers in national and international journals. His research areas include Data mining and Network security.

K. Rameshkumar is presently doing Ph.D Fulltime at Dept of CSE, Alagappa university, Karaikudi. He has more than 3 years of industrial experience. He has participated and presented the research papers in various national and International level conferences. His research interests are Data mining, E-security and E-learning.

N. Venkatesan is working as Assistant Professor & Head in Information Technology Department in Bharathiyar college of Engineering and Technology, Karaikal, Pondichery. He holds Bachelor’s degree in mathematics, Master’s degree in computer applications from Bharathidasan University, MTech from Allagabad Agricultural University, and MPhil in Comp. Sci. Currently doing Ph.D. He has been member of ISTE. He published 9 papers in National and International conferences. He published a book data mining and warehousing. He is guiding Engineering and MCA students’ projects. He is guiding M.Phil. Scholars.