Improving Maximal Frequent Item set Mining for Sparse Dataset

B. sujatha Asst.prof, Ramesh babu Varugu Asst.prof

ABSTRACT

Mining of maximal frequent patterns is a basic problem in data mining applications. Small and useful association rules can be generated from maximal frequent itemset. The algorithms which are used to generate the maximal frequent patterns must perform efficiently. Most of the existing algorithms passed all frequent itemsets as candidates to the recursive algorithm which generates MFI. But the sparse dataset has huge number of frequent items and each frequent item has very small number of candidate items. This paper presents FastMFIMiner algorithm to generate MFI quickly from sparse dataset. It works efficiently even when the number of itemsets is more. The proposed algorithm has been compared with GenMax, Mafia and DepthProject for sparse and mushroom dataset and the results show that the proposed algorithm generates maximal frequent patterns quickly than existing algorithms.

Keywords:  Maximal Frequent Patterns, Sparse Dataset, Mining MFIs

1. INTRODUCTION

A fundamental problem for mining association rules [5] is to mine frequent itemsets (FI’s). In a transaction database, if we know the support of all frequent itemsets, the association rules generation is straightforward. However, when a transaction database contains large number of large frequent itemsets, mining all frequent itemsets might not be a good idea. The drawback of mining all frequent itemsets is that if there is a large frequent itemset with size n then almost all $2^n$ candidate subsets of the items might be generated. However, since frequent itemsets are upward closed, it is sufficient to discover only all maximal frequent itemsets (MFI’s). Thus, a lot of work is focused on discovering only all the maximal frequent itemsets (MFIs).

There is much research on methods for generating all frequent itemsets efficiently [6, 7, 8] or just the set of maximal frequent itemsets [1, 2, 3, 4]. When the frequent patterns are long (more than 15 to 20 items), FI and even FCI become very large and most traditional methods count too many itemsets to be feasible. Straight Apriori-based algorithms count all of the 2 subsets of each k-itemset they discover, and thus do not scale for long itemsets. Other methods use “lookaheads” to reduce the number of itemsets to be counted. However, most of these algorithms use a breadth-first approach, i.e. Finding all k-itemsets before considering (k+1) itemsets. This approach limits the effectiveness of the lookaheads, since useful longer frequent patterns have not yet been discovered. Then, the merits of a depth-first approach have been recognized. The database representation is also an important factor in the efficiency of generating and counting itemsets.

Generating the itemset $Z = (X \cup Y)$ refers to creating $t(Z) = t(X) \cap t(Y)$, and counting is the process of determining $\text{support}(Z)$ in $T$. Most previous algorithms use a horizontal row layout, with the database organized as a set of rows and each row representing a transaction. The alternative vertical column layout associates with each item $X$ a set of transaction identifiers (tids) for the set $t(X)$. The vertical representation allows simple and efficient support counting.

2. RELATED WORKS
Max Miner [1] is an algorithm introduced by Roberto Bayardo for finding the maximal frequent patterns. It uses efficient pruning techniques such as item reordering to quickly narrow the search. It introduces Support lower bound computation method for frequency computations. Max Miner employs a breadth first traversal of set enumeration tree of itemset. It reduces database scanning by employing a look ahead pruning strategy.

GenMax is a backtrack search based algorithm introduced by K. Gouda and M.J.Zaki [3] for mining maximal frequent itemsets. GenMax uses a vertical database format, where data is represented in item-tidset format. GenMax uses a number of optimizations to prune the search space. It introduces new techniques such as progressive focusing to perform fast superset checking, reordering for search space pruning and diffset propagation to perform fast frequency computation.

Depth Project [2] finds long itemsets using a depth first search of a lexicographic tree of itemsets, Depth project uses a bitstring representation of database and counting method based on transaction projections along its branches. Bucketing technique is used to improve the counting times. It returns superset of the MFI and requires post-pruning to eliminate non-maximal item sets.

Mafia [5] is one of the recent methods for mining the maximal frequent patterns. In Mafia the Search strategy combines a vertical bitmap representation of the database with an efficient relative bitmap compression schema. Mafia uses three pruning strategies to remove non-maximal sets. The first one is the look-ahead pruning introduced in MaxMiner. The second technique checks if \( t(X) \subseteq t(Y) \). If so X is considered together with Y for extension. The last method is to check if any existing maximal set includes the new set. Mafia requires a post-pruning step to eliminate non-maximal patterns.

3. PROPOSED WORK

When the MFI mining algorithms recursively construct many candidates, the performances of these approaches degraded, if the database is massive or the threshold for mining frequent patterns is low. Most of the existing algorithms take all frequent items as candidates from which all MFIs are generated. Instead of passing huge number of frequent items as candidates, here the recursive algorithm is invoked by every frequent item and its candidate pair because of frequent items having less number of candidates.

The main idea of the approach is to generate MFI quickly from the sparse dataset. In sparse dataset, there is lot of frequent itemset and each frequent itemset have less number of candidates. The maximal frequent itemset cardinality is not much smaller than frequent itemsets. The mean pattern length is also low. So instead of passing all frequent items as candidates, each frequent item and its candidates are passed to MineMFI algorithm. Tidset of frequent item is also sent, so that it is easy to compute the frequency of an itemsets. MineMFI algorithm is called for each frequent item and its candidate set pair. Once MineMFI is invoked, all MFI that include the particular frequent item are obtained.

The first step of FastMFIMiner is extracting all frequent items and reordering the frequent items in ascending order of their support. In second step, candidate items (CIi) for each frequent item (FIi) are generated and candidate items are arranged in increasing order of their support. In second step, candidate items (CIi) for each frequent itemset before finding all frequent itemsets.

FastMFIMiner uses backtracking method to mine MFI and backtrack search space can be smaller than the full space because of using generating candidate and proceed method. The FastMFIMiner generates candidates, once a frequent extension is obtained and generates maximal frequent itemset before finding all frequent itemsets.

The MineMFI method is invoked number of times which is less than or equal to the number of frequent item in the dataset. The FastMFIMiner method is not invoked, when the combination of FIi and candidate items of FIi is frequent. For sparse dataset this early finding of MFI has improved performance than other existing algorithms.

FastMFIMiner is explained with the following example. Let us consider the transaction database \( d \) which includes five different items, \( I = \{ A, B, C, D, E \} \) and six transactions \( T = \{ 1, 2, 3, 4, 5, 6 \} \). The vertical data format of the database d is given in Table 1. Support of an item is number of transactions that include the item. All frequent items are extracted and reordered in ascending order with respect to the support. The support is directly given by the number of transactions in the tidset of each item. For example, consider the minimum support to be 3 transactions. In database d, all items are having more than two tids in the tidset, all items are frequent. The items A, B, C, D and E are reordered in ascending order with respect to the support and these are considered to next level. The frequent items are A, B, C, E and D.
The next frequent item and its candidate set is C and {E, and added to MFI. The next frequent item and its candidate set {B, D, E} and {B, C, D}.

Candatate sets for each frequent item is generated in the next step. All candidates are reordered in increasing order of support. The candidates of frequent item B is \{B, C, D\}, frequent item C is \{C, E, D\}, frequent item C is \{E, D\}, frequent item E is \{D\} and frequent item D is \{\}. The first frequent item and its candidate set is A and \{B, C, D\} respectively. For this pair, FI \cup CI has no superset in MFI and the itemset \{A, B, C, D\} is frequent and added to MFI. The next frequent item and its candidate set is B and \{C, E, D\} respectively. For this pair, FI \cup CI has no superset in MFI and the itemset \{B, C, D, E\} is frequent and added to MFI. The subsequent pairs are E \cup \{D\} and D \cup \{\} having superset in MFI, ignored. MFI with support 3 transactions returned by FastMFIMiner are \{A, B, C, D\} and \{B, C, D, E\}.

Let us now consider the minimum support to be 4. In database d, all items except A are having more than three tids in the tidset, all items are frequent. The frequent items are B, C, D and E and are reordered in ascending order with respect to the support and these are considered to next level. Frequent items with support 4 in database d are B, C, E and D.

 Candidate sets for each frequent item is generated in the next step. All candidates are reordered in increasing order of support. The candidates of frequent item B is \{C, D, E\}, frequent item C is \{E, D\}, frequent item E is \{D\} and frequent item D is \{\}.

The first frequent item and its candidate set is B and \{C, E, D\} respectively. For this pair, FI \cup CI has no superset in MFI and the itemset \{B, C, E, D\} is not frequent. MineMFI method is called and the frequent item B, candidates of B (C, E, D) and tidset of B are passed to mine MFI which generates all MFI that includes the item B using backtracking method. MineMFI method returns the maximal frequent itemsets \{B, D, E\} and \{B, C, D\}.

The next frequent item and its candidate set is C and \{E, D\} respectively. For this pair, FI \cup CI has no superset in MFI and the itemset \{C, E, D\} is frequent and is added to MFI.

The subsequent pairs are E \cup \{D\} and D \cup \{\} have superset in MFI and are ignored. MFI with support count 4 in database d [Table 1] are \{B, C, D\} and \{B, D, E\} and \{C, E, D\}.

**Table 1.** Vertical item-tidset of database D.

<table>
<thead>
<tr>
<th>Item</th>
<th>Tidset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T1, T3, T5</td>
</tr>
<tr>
<td>B</td>
<td>T1, T3, T4, T5, T6</td>
</tr>
<tr>
<td>C</td>
<td>T1, T2, T3, T4, T5</td>
</tr>
<tr>
<td>D</td>
<td>T1, T2, T3, T4, T5, T6</td>
</tr>
<tr>
<td>E</td>
<td>T1, T2, T4, T5, T6</td>
</tr>
</tbody>
</table>

In the next level candidate items for each frequent item is obtained. All candidates are reordered in increasing order of support. The candidates of frequent item A is \{B, C, D\}, frequent item B is \{C, E, D\}, frequent item C is \{E, D\}, frequent item E is \{D\} and frequent item D is \{\}.

The first frequent item and its candidate set is A and \{B, C, D\} respectively. For this pair, FI \cup CI has no superset in MFI and the itemset \{A, B, C, D\} is frequent and added to MFI. The next frequent item and its candidate set is B and \{C, E, D\} respectively. For this pair, FI \cup CI has no superset in MFI and the itemset \{B, C, D\} is frequent and added to MFI. The subsequent pairs are E \cup \{D\} and D \cup \{\} respectively. For this pair, FI \cup CI has no superset in MFI and the itemset \{B, C, D, E\} is frequent and added to MFI.

The first element in candidate of FI i (CIi) is combined to FI i and new candidates are generated. This process is repeated until there is no candidate for frequent itemset. In case of sparse datasets, data contains small patterns and there is not much repetition of patterns in dataset. FastMFIMiner has better performance than the existing algorithms.

**Algorithm – FastMFIMiner**

**Input:** dataset D, support S

<table>
<thead>
<tr>
<th>Tid</th>
<th>Items</th>
<th>Maximal Frequent Itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABCDE</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CDE</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ABCD</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>BCDE</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ABCDE</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>BDE</td>
<td></td>
</tr>
</tbody>
</table>

The maximal frequent itemsets of database d [Table 1] for support 4 and 3 transaction is given in Table 2. MFI miner mines maximal frequent itemsets using depth first search strategy, generates candidates whenever a frequent itemset is generated. This process is repeated until there is no candidate for frequent itemset. In case of sparse datasets, data contains small patterns and there is not much repetition of patterns in dataset. FastMFIMiner has better performance than the existing algorithms.

**Definition 1:** FI denotes the set of frequent item and CI denotes the set of candidate items. FI denotes a frequent item and CI denotes the candidate items of frequent item FI. Each time the first element in candidate of FI (CI) is combined to FI and new candidates are generated for each frequent extension using generateCandidate method. Once the element is combined to FI, it is removed from candidates of FI (CI).

**Definition 2:** Candidates of FI (CI) are generated from frequent items FI by removing FI. For example, frequent items are \{A, B, C, D\}. For frequent item A, the candidates are \{B, C, D\}. For frequent item B, the candidates are \{C, D\}. For frequent item C the candidates is \{D\} for frequent item D the candidates is \{\}.

The frequent itemset and its new candidate pair is added to MFI, if it has no superset in MFI and is frequent. Otherwise the next element in candidate of FI (CI) is combined to the FI and new candidates are generated.

**Definition 3:** MineMFI method is not called, if FI \cup CI is frequent and has no superset in MFI or it has superset in MFI.

<table>
<thead>
<tr>
<th>Tid</th>
<th>Items</th>
<th>Maximal Frequent Itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABCD</td>
<td>BCD</td>
</tr>
<tr>
<td>2</td>
<td>BCDE</td>
<td>BDE</td>
</tr>
<tr>
<td>3</td>
<td>ABCD</td>
<td>CDE</td>
</tr>
</tbody>
</table>

**Definition 3:** MineMFI method is not called, if FI \cup CI is frequent and has no superset in MFI or it has superset in MFI.
Output: Maximal Frequent Itemsets MFI
FastMFIMiner (Dataset D, Support S) BEGIN
1. Generate frequent items and reorder them in ascending order of their support.
2. Generate candidate items for each frequent item and reorder the candidates in increasing order of the support.
3. For each \( x \in \text{FIs} \)
   a. If \( x \cup \text{candidates}(x) \) has no superset in MFI
      i. if size of \( x \) is 1 or 2 then add \( x \) to MFI
      // most of the sparse dataset the candidate items of frequent item may be 1 or 2;
      ii. else if \( x \cup \text{candidates}(x) \) is frequent then add \( x \) to MFI;
      iii. else \( \text{MineMFI}(x, \text{candidates}(x), \text{tid}(x)) \)
END

MineMFI
Input: frequent item, candidate set, Tidset of the frequent item.
Output: Maximal Frequent Itemsets that includes the frequent item.
MineMFI (frequent item, candidateset, ftids) BEGIN
For each \( x \in \text{candidateset} \)
If(frequent \( \cup \text{candidateset} \) has superset in MFI then return;
\( N\text{frequent} = \text{frequent} \cup x \) // current frequent itemset
\( N\text{tids} = x.\text{tid} \cap \text{ftids} \) // Tidset of current frequent itemset
\( \text{candidateset.remove}(x) \) // candidates of current frequent itemset
if candidateset is empty \&\& \( N\text{frequent} \) has no superset in MFI
   \( \text{MFI.add}(N\text{frequent}); \text{return}; \)
   newcandidate = generatecandidates(Nfrequent,candidateset,Ntids) //exact candidate of current frequent itemset
   if newcandidate is empty
      if Nfrequent has no superset in MFI
         MFI.add(Nfrequent);
      else
         generateMFI (Nfrequent, newcandidate,Ntids)
   END
END

Generate Candidates
Input: Frequent Itemset, Candidate set, Tidset of the Frequent Itemset.
Output: Exact Candidates of Frequent Itemset.
Generate Candidates (frequent, candidate, ftids)
BEGIN
cand=null; // cand will contain the exact candidates of frequent item(frequent)
for each \( x \in \text{candidate} \)
If(ftids \( \cap \text{tid}(x) \geq \text{support} \)
   cand.add(x); // candidates are stored in increasing order of support.
return (cand);
END

The MineMFI method follows depth first search strategy and candidate generation method. Using depth first search maximal frequent itemsets can be mined before generating all frequent itemsets. In breath first search method, all \( I_{l+1} \) - frequent itemsets are obtained, after obtaining all \( I_l \) - frequent itemsets. Tidsets of each frequent itemset is obtained and passed to the candidategenerate method. Once the tidset of a frequent itemset is obtained, the candidates of frequent extensions are obtained easily from the tidset of frequent itemset. This process reduces the frequency computation time.

Pruning
Most of the standard algorithm like mafia [4], depthproject[2], genmax[3] get all frequent items as candidates and MFI are obtained from these candidates. To generate the MFI quickly, FastMFIMiner uses two pruning techniques. The first pruning technique is reordering of items with respect to its support in ascending order. This reordering technique is introduced by Roberto Bayardo in MaxMiner algorithm [1] for mining maximal frequent itemsets. Once frequent itemset are generated, they are reordered with respect to its support in ascending order. The second pruning technique is the recursive method MineMFI is not called, if the combination of frequent item (FIi) and candidates of FIi (CIi) (FIi \( \cup \) CIi) is frequent.

Performance Evaluation
The performance of FastMFIMiner algorithm is compared with three different algorithms, it is observed that the performance is varied significantly depending on the dataset characteristics. To evaluate the performance of FastMFIMiner algorithm, four different benchmark datasets is used. All these datasets are downloaded from FIMI Repository [9]. Dataset taken for experiment are T10I4D100K, T40I10D100K, Retail and Mushroom Dataset.

The first dataset is T10I4D100K which contains 1000 attributes and 100,000 records. The average record length is 10. The mean pattern length is very small and it is around 2 to 3. In T10I4D100K dataset, the number of frequent items is huge and frequent itemset will have small number of candidates. The performance of FastMFIMiner algorithm has been compared with
GenMax[3], Mafia[4], and Depth Project[2] algorithm for various support and results show that FastMFIMiner gives superior performance than the existing algorithms. Figure 1 illustrates that, the FastMFIMiner algorithm has better performance, when compared to conventional GenMax, Mafia and Depth Project algorithm.

![T10I4D100K Dataset](image)

**Fig.1 Performance of FastMFIMiner algorithm on T10I4D100K dataset.**

The second dataset is T40I10D100K which contains 1000 attributes and 100,000 records. The average record length is 40. In T40I10D100K dataset, the number of frequent items is huge and frequent itemset will have small number of candidates. The mean pattern length is very small and it is around 2 to 6. Mean pattern length is slightly greater than T10I4D100K dataset. Number of maximal frequent itemset is not much smaller than the number of all frequent patterns. Figure 2 illustrates that, the FastMFIMiner algorithm has better performance than GenMax, Mafia and Depth Project algorithms on T40I10D100K dataset.

![T40I10D100K](image)

**Fig.2 Performance of FastMFIMiner algorithm on T40I10D100K dataset.**

The third dataset is Retail which contains 16,470 items and 88,162 transaction. The average number of distinct items in each transaction is 13 and most transaction contains items between 7 and 11 items. Mean pattern length is also very short. The maximum number of maximal patterns is of length one or two. Number of maximal frequent itemsets is not much smaller than the number of all frequent itemsets. In retail dataset, the number of frequent items is huge and frequent itemset will have small number of candidates. Figure 3 illustrates that, the FastMFIMiner algorithm has better performance than GenMax and Mafia and Depth Project algorithms on Retail dataset.

![Retail Dataset](image)

**Fig.3 Performance of FastMFIMiner algorithm on Retail dataset.**

Mushroom dataset contains 120 items and 8,124 transactions. The average transaction length for mushroom is 23 thus a maximal pattern spans almost a full transaction. The total number of maximal frequent itemset is about 1000 times smaller than all frequent itemsets [3]. On mushroom dataset, the results of FastMFIMiner algorithm are similar to Mafia. FastMFIMiner algorithm has better performance than GenMax. Because in mushroom dataset a smaller itemset have many number of maximal frequent itemsets. To perform maximality checking one has to test against a large set of maximal itemsets. GenMax has better performance than Depthproject. Figure 4 illustrates that, the FastMFIMiner algorithm has better performance than GenMax and Depth Project algorithms on Mushroom dataset.

![MUSHROOM](image)

**Fig. 4 performance of FastMFIMiner algorithm on Mushroom dataset.**

4. **CONCLUSION**

In this paper we have introduced FastMFIMiner algorithm for mining maximal frequent patterns quickly...
from sparse dataset. The initial candidates for every frequent item are generated by finding association among frequent items. The generated candidates are sorted in increasing order of their support. The frequent item $U$ its candidate set are added to MFI directly, if the $\text{FI}_U \cup \text{CI}_i$ is frequent and has no superset in MFI. Otherwise MineMFI is invoked to obtain MFI from frequent items $\text{FI}_U$ and candidate set $\text{CI}_i$. The performance of FastMFI Miner is tested using benchmark dataset and results show that, FastMFI Miner generates MFI very quickly from sparse dataset.

REFERENCES


FIMI 2004- Frequent itemset Mining Implementations URLs
i) http://www.adrem.ua.ac.be/~goethals/
ii) http://fimi.cs.helsinki.fi/fimi04/
iii) http://icdm-04.cs.uni-dortmund.de/