Surveillance of Efficient Algorithms for Mining Frequent Itemsets and Closed frequent itemsets

Mrs.K.Jayavani ¹, Dr.G.M.Kadhar Nawaz ²
¹ Research Scholar
Manonmaniam Sundaranar University, Tirunelveli
² Director, Dept.Of.Computer Application
Sona College of Technology, Salem

Abstract
In a transaction database, a frequent itemset is an itemset included in at least a specified number of transactions. Frequent itemset generation is the prerequisite and most time-consuming process for association rule mining. Many researchers invented ideas to generate the frequent itemsets. The time required for generating frequent itemsets plays an important role. Some algorithms are designed, considering only the time factor. In this paper we will deeply discuss the analysis of algorithms such as Apriori, FP-growth for frequent itemset generation and Closet, Charm for closed frequent itemset generation on theoretical basis and also analyse the unifying features of algorithms.

Keywords—Frequent Itemset, Closed Frequent Itemset, Apriori, FP-Growth, Closet, Charm.

1. Introduction
In recent years the size of the database has increased rapidly. This has led to a growing interest in the development of tools that are capable in the automatic extraction of knowledge from data. The term data mining or knowledge discovery in database has been adopted for a field of research dealing with the automatic discovery of implicit information or knowledge within the databases. The implicit information within databases, mainly the interesting association relationships among sets of objects that lead to association rules may disclose useful patterns for decision support, financial forecast, marketing policies, even medical diagnosis and many other applications. Frequent pattern mining plays an essential role in many important data mining tasks from database such as association rules, correlations, sequences, classifiers, clusters and many more. From these analyzing tasks association rule is one of the most popular problems. Association rule explains how often items are purchased together in a market basket analysis. However it is also well known that frequent pattern mining often generates a very large number of frequent itemsets and rules. These rules are useful for decisions concerning product pricing, promotions, store layout and many others. Here, we analyze the efficient
algorithms called Apriori, FP-growth and Closet, Charm to generate frequent itemsets and closed frequent itemsets respectively.

2. Problem study
Frequent itemset mining takes an efficient role and is acknowledged in the data mining field because of its broad application in mining, includes market analysis, genome analysis and drug analysis (molecular fragment mining). Efficient algorithm for mining frequent itemsets are crucial for mining association rule as well as for many other data mining tasks. The major challenge found in frequent pattern mining is a large number of result patterns. As the minimum threshold becomes lower, an exponentially large number of itemsets are generated.

Therefore, pruning unimportant patterns can be done effectively in mining process and that becomes one of the main topics in frequent pattern mining. Consequently, the main aim of this paper is to provide the analysis of various algorithms called Apriori, FP-growth, Closet and Charm algorithms for extraction of frequent itemsets and closed frequent itemsets from transactional databases. The analysis includes time factor for executing the algorithms, performances, efficiency and scalability of the algorithm.

3. Related work

3.1 Apriori algorithm
R. Agrawal et al in 1993 proposed this famous Apriori algorithm, an influential algorithm for mining frequent itemsets for Boolean association rule over transactional database. This algorithm is used to discover knowledge for the purpose of explaining current behavior, predicting future outcomes and provide support for banks decision making processes and also for some other Business Intelligence purposes. Market Basket Analysis is the best example for the association rule mining, finding the items that are purchased together more frequently than others.

3.1.1 Basic concepts

Frequent Itemsets (Lk): The sets of item which has minimum support.
Candidate Itemset (Ck): The itemset that require testing to see if they fit in to certain requirements.
Support: Interesting association rule can be measured with the help of support criteria which defines the percentage of transaction that contains both sides of implications in the association rule.
Support (k) = count (k)/total transaction.
Minimum support: Threshold for support. Those values are assigned by the user. This helps to eliminate the non-frequent items from the database.
Steps,
- First, the set of candidate-1 itemset is found (C1).
- Then support is calculated by counting the occurrence of the item in transactional database.
- After that we will prune the C1 using minimum support Criteria. The item which satisfies the minimum support criteria is taken into consideration for the next process and which is known as L1.
- Then again candidate set generation is carried out and the 2-itemset which is generated known as C2.
Again we will calculate the support of the 2-Itemset (C2). And we will prune C2 using Minimum support and generate L2.

This Process Continues till there is no Candidate set and frequent itemsets can be generated.

3.1.2 Pseudo Code

\( C_k \): Candidate itemsets of size \( k \)

\( L_k \): frequent itemset of size \( k \)

\( L_k = \{ \text{frequent items} \} \);

For (\( k = 1; L_k = \emptyset; k++ \)) do begin

\( C_{k+1} \): candidates generated from \( L_k \);

For each transaction \( t \) in database do

Increment the count of all candidates in \( C_{k+1} \) that are contained in \( t \)

\( L_{k+1} \): candidates in \( C_{k+1} \) with min_support

end

return \( C_K \ L_K \);

3.1.3 Analysis

Apriori uses a bottom up approach, where frequent subsets are extended one item at a time (a step known as candidate generation), and groups of candidates are listed against the data. It follows breadth first search method and it uses hash tree to store frequent itemsets and candidate frequent itemsets. Because of the download closure property of the frequency pattern, only candidate frequent itemsets, whose subsets are all frequent, are generated in each database scan. Candidate frequent itemset generation and subset testing are all based on the hash trees. We can also say that this algorithm uses level-wise search. Unlike other mining algorithms, Apriori is a very simple structure.

Since Apriori follows candidate set generation and test approach, it is costly when there exist long patterns and also very tedious to repeatedly scan the database and memory consumption also high. Here the transactions are not stored in main memory. This assumes transaction database as memory resident and also requires many database scans. Even though Apriori uses itemset property, easily parallelized and easy to implement, it has several limitations like scanning time, memory optimization and candidate generation.

The performance of the Apriori algorithm may degrade significantly for dense datasets because of the increasing width of transactions. Most of the drawbacks in Apriori algorithm can be solved by several improved Apriori approaches.

3.2 FP-Growth algorithm

FP-Growth, a novel algorithm is proposed by Han et al. This is a depth first algorithms. In this method a data structure called the FP-tree is used for storing frequency information of the original database in a compressed form. Only two database scans are needed for the algorithm and also it does not generate candidate itemset. Every branch of the FP-tree represents a frequent itemset and the nodes along the branches are stored in decreased order of frequency of the corresponding items with leaves representing the least frequent items. Compression is achieved by building the tree in such a way that overlapping itemsets share prefixes of the corresponding branches. It has two approaches.

**Step 1**: Build a compact data structure called the FP-tree.
Step 2: Extracts frequently itemsets directly from the FP–tree.
In simple words, this algorithm works as follows: first it compresses the input database creating an
FP-tree instance to represent frequent items. After this first step it divides the compressed database
into a set of conditional databases, each one associated with one frequent pattern. Finally, each such
database is mined separately. A number of optimization are used for reducing time and space of the
algorithm.

Algorithm 1: FP-Tree Construction

FP-Tree is constructed using 2 passes over the data-set:

Pass 1:
- Scan data and find support for each item.
- Discard infrequent items.
- Sort frequent items in decreasing order based on their support

Pass 2:
- First, create the root of the tree, labeled with “null”
- Scan the database D a second time. (First time we scanned it to create 1-itemset and then L).
- The items in each transaction are processed in L order (i.e. sorted order).
- A branch is created for each transaction with items having their support count separated by
  colon.
- Whenever the same node is encountered in another transaction, we just increment the support
  count of the common node or Prefix.
- To facilitate tree traversal, an item header table is built so that each item points to its
  occurrences in the tree via a chain of node-links.
- Now, the problem of mining frequent patterns in database is transformed to that of mining the
  FP-Tree.

Algorithm 2: FP-growth (Frequent itemset generation from FP-tree)

Input: FP-tree constructed based on Algorithm 1, using DB
and a minimm support threshold $\varepsilon$

Output: The complete set of frequent patterns.

Method: Call FP-growth (FP-tree, null), which is implement
ed as follows:

Procedure FP-growth (Tree, $\alpha$)
{
(1) IF Tree contains a single path $P$
(2) THEN FOR EACH combination (denoted as $\beta$) of
  the nodes in the path $P$ DO
(3) generate pattern $\beta \cup \alpha$ with support = minimum
Analysis

FP-growth is an interesting algorithm because it illustrates how a compact representation of the transaction data set, FP-tree helps to efficiently generate frequent itemsets. This compact FP-tree is usually substantially smaller than the original database and thus saves the costly database scans in the subsequent mining process. This algorithm generates frequent itemsets from an FP-tree by exploring the tree in a bottom up fashion that is from the leaves towards the root. This algorithm does not follow candidate – generation- test and hence cost is dramatically reduced. Even though FP-tree is expensive and takes time to build, but once it is built, frequent itemsets are read off easily.

This algorithm stores data, FP-tree in memory rather than disk if the size of the FP-tree is small. This will allow us to extract frequent itemsets directly from the structure in main memory instead of making repeated passes over the data stored on disk. FP-tree may not fit in to the memory if the size of the FP-tree is large and hence disk based FP-Tree approach should be carried out. FP-tree runs faster than Apriori in circumstances but it is not inherently faster than Apriori.

Many research experiments shows that FP-growth outperforms Apriori by several orders of magnitude but FP-growth uses more complicated data structure and mining techniques. It follows Divide and conquer approach and the size of the tree is bounded by occurrences of frequent items. The height of the tree is bounded by the maximum number of items in a transaction.

3.3 Closet

The Closet algorithm is proposed by Pei et al in 2003. This algorithm is used for mining closed itemsets with the development of three techniques:

- Applying a compressed, frequent pattern tree FP-tree structure for mining closed itemsets without candidate generation,
- Developing a single prefix path compression technique to identify frequent closed itemsets and,
- Exploring a partition-based projection mechanism for mining in large databases.

It performs a depth-first search, that is after discovering a frequent itemset A, it searches for super patterns of A before checking A’s siblings.

3.3.1 Mining process of Closet

- Find frequent items. Scan TDB (Transactional database) to find the set of frequent items and derive a (global) frequent itemsets called f_list where the items are sorted in support descending order.
Divide search space. All the frequent closed itemsets can be divided into a number of non-overlapping subsets based on the $f_\text{list}$. Once all subsets are found, the complete set of frequent closed itemsets is done.

Find subsets of frequent closed itemsets. The subsets of frequent closed itemsets can be mined by constructing corresponding conditional databases and mine each recursively.

The search for closed itemsets can be improved further by a few optimization techniques as shown below.

- **Optimization 1**: Compress transactional and conditional databases using an FP-tree structure.
- **Optimization 2**: Extract items appearing in every transaction of conditional database.
- **Optimization 3**: Directly extract frequent closed itemsets from FP-tree.
- **Optimization 4**: Prune search branches.

**Algorithm 3: Mining frequent closed itemsets by the FP-tree method.**

**Input**: Transaction database TDB and support threshold min sup

**Output**: The complete set of frequent closed itemsets;

**Method**:

1. Initialization. Let $FCI$ be the set of frequent closed itemset. Initialize $FCI \leftarrow \emptyset$
2. Find frequent items. Scan transaction database TDB, compute frequent item list $f_\text{list}$;

**Subroutine CLOSET(X, DB, f_\text{list}, FCI)**

**Parameters**:
- $X$: the frequent itemset if $DB$ is an $X$-conditional database, or $\emptyset$ if $DB$ is TDB;
- $DB$: transaction database of conditional database;
- $f_\text{list}$: frequent item list of $DB$;
- $FCI$: The set of frequent closed itemsets already found.

**Method**:

1. Let $Y$ be the set of items in $f_\text{list}$ such that they appear in every transaction of $DB$, insert $X \cup Y$ to $FCI$ if it is not a proper subset of some itemset in $FCI$ with the same support; // Applying Optimization 2
2. Build FP-tree for $DB$, items already be extracted should be excluded; // Applying Optimization 1
3. Apply Optimization 3 to extract frequent closed itemsets if it is possible;
4. Form conditional database for every remaining item in $f_\text{list}$, at the same time, compute local frequent item lists for these conditional databases;
5. For each remaining item $i$ in $f_\text{list}$, starting from the last one, call $CLOSET(iX, DB_{ji}, f_\text{list}_i, FCI)$ if $iX$ is not a subset of any frequent closed itemset already found with the same support count, where $DB_{ji}$ is the $i$-conditional database with respect to $DB$ and $f_\text{list}_i$ is the corresponding frequent item list. // Applying Optimization 4.

**Analysis**

ClosedSet uses a novel frequent pattern (FP-tree) structure which is a compressed representation of all the transactions in the database for mining closed itemsets without candidate generation. We can also say
this is an extension of FP-growth algorithm. Follows depth first search methods. This proposes a partition based approach, which can reduce the space cost dramatically. Uses divide and search method to divide the found frequent closed itemsets into number of overlap subsets

Closet develops a single prefix path compression technique to identify closed itemsets quickly. It is a memory resident when the size of the FP-tree is small. When the transaction database is large then, it is unrealistic to construct a main memory based FP-tree. In such a case disk-based FP-trees are constructed. This algorithm uses many optimization techniques to reduce time, memory consumption and also to improve its efficiency

3.4 Charm
Charm is an algorithm proposed by Mohammed J.Zaki and Ching-Jui Hsiao in 2005. It is used to generate closed frequent itemsets. An itemset is closed in a data set if there exists no superset that has the same support count as this original itemset. Charm is unique in that it simultaneously explores both the itemset space and transaction space, over a novel IT-tree (itemset-tidset tree) search space. This algorithm uses a hybrid search method that skips many levels of the IT-tree to quickly identify the frequent closed itemsets, instead of having to enumerate many possible subsets. Charm also utilizes a novel vertical data representation called diffset for fast frequency computations.

3.4.1 Mining process of Charm
The algorithm starts by initializing the set of nodes to be examined to the frequent single items and their tidsets in Line 1. The main computation is performed in CHARM-ET which returns the set of closed frequent itemsets C.

CHARM-ET is responsible for testing each branch for viability. It extracts each itemset-tidset pair in the current node set Nodes (X_i × t(X_i), Line 3), and combines it with the other pairs that come after it (X_j × t(X_j), Line 5) according to the total order f. The combination of the two itemset-tidset pairs is computed in Line 6. This tests the resulting set for required support and may modify the current node set by deleting itemset-tidset pairs that are already contained in other pairs. It also inserts the newly generated children frequent pairs in the set of new nodes New N.

If this set is non-empty we recursively process it in depth-first manner (Line 8). We then insert the possibly extended itemset X, of X_i, in the set of closed itemsets, since it cannot be processed further; at this stage any closed itemset containing X_i has already been generated. We then return to Line 3 to process the next (unpruned) branch.

Charm algorithm
CHARM (δ, ⊆ I× T, min sup):
1. Nodes = \{ I_j × t(I_j) : I_j ∈ T ∧ |t(I_j)| ≥ min sup\}
2. CHARM-ET (Nodes, C)
3. return C //all closed sets

CHARM-ET (Nodes, C):
4. for each X_i × t(X_i) in Nodes
5. NewN= ∅ and X=X_i
6. for each X_j × t(X_j) in Nodes, with f(j) ≥ f(i)
7. X = X ∪ X_j and Y = t(X_j) ∩ t(X_i)
8. CHARM-Property(Nodes,NewN)
9. if (NewN = ∅) then CHARM-ET (NewN)
10. C=C∪X// if X is not subsumed.

CHARM-Property (Nodes,NewN):
11. if (|Y| ≥ minsup) then
12. if t(X) = t(Xj ) then //Property 1
13. Remove Xj from Nodes
14. Replace all Xj with X
15. else if t(X) ⊂ t(Xj ) then //Property 2
16. Replace all Xi with X
17. else if t(X) ⊃ t(Xj ) then //Property 3
18. Remove Xj from Nodes
19. Add X × Y to NewN //use ordering f
20. else if t(X) ≠ t(Xj ) then // property 4
21. Add X × Y to NewN.

Analysis
Instead of mining all frequent itemsets, Charm considers only to mine a closed frequent itemsets which is much smaller than the set of all frequent itemsets. So that many redundant rules can be eliminated. Charm is unique in that it simultaneously explores both the itemset space and transaction space, unlike other algorithms like Apriori, FP-growth which only exploits the itemset search space. And also Charm avoids enumerating all possible subsets of a closed itemset when enumerating the closed frequent itemsets. This algorithm uses a highly efficient hybrid search method that skips many levels of the IT-tree to quickly identify the frequent closed itemsets. The performance of this algorithm is good with dense datasets such as telecommunications and census data, where there are many, long frequent patterns. The use of a technique called diffsets drastically cut down the size of memory required to store intermediate results. Thus the entire working set of patterns can fit in to main memory even for large databases.

3.5 Comparison
Apriori and FP-growth algorithms are considered as an efficient algorithm for mining frequent itemsets. Apriori is a horizontal formatting method and it follows bottom–up breadth first search approach with candidate itemset generation. Apriori shows good performance with sparse dataset such as market basket data, where the frequent patterns are shorter. Performance of this algorithm degrades incredibly with dense datasets. That is, this algorithm is not suitable for long pattern mining. FP-growth, Closet and Charm are the algorithms designed for mining long patterns. Compared with breadth-first algorithms such as Apriori and its variants, which may need as many database scans as the length of the longest pattern, FP-growth method only needs two database scans. FP-growth does not generate candidate itemset and it is based on partition-based, divide and conquer methods. It enumerates compressed database-FP tree structure and has no repeated scan of entire database. Hence when compared to Apriori, FP-growth is the good choice for mining frequent itemsets especially for long pattern mining. Execution time of FP-growth is smaller than Apriori, since time is wasted by producing candidates every time. Due to compact structure and has no candidate generation. FP-growth requires less memory when compared to Apriori. FP-growth outperforms the standard Apriori
algorithm by several orders of magnitude. The run time performance of FP-growth depends on the compact factor of the dataset. If the resulting conditional FP-trees are very bushy, then the performance of the algorithm degrades significantly because it has to generate a large number of sub programs and merge the results returned by each subprogram. To overcome this problem, the algorithms called Closet and Charm are used.

In Closet, the method of FP-growth is extended. Both mines only closed frequent itemsets and their corresponding association rule. Hence it will substantially reduce redundant rules to be generated. Closet uses a novel frequent pattern tree (FP structure) which is a compressed representation of all the transactions in the database. It follows divide and conquer and database projection approach to mine long patterns. This performs a depth-first search strategy. Similar to Closet, Charm also searches for patterns in a depth first manner. Charm uses a novel Itemset-Tidset (IT-tree) search space. The difference between Charm and Closet is that Charm stores the data set in a vertical format where a list of row IDs is maintained for each dimension. These row ID lists are then merged during a “column enumeration” procedure that generates row ID lists for other nodes in the enumeration tree. In addition a technique called diffset is used to reduce the length of the row lists as well as the computational complexity of merging them. When compared to FP-growth, Closet and Charm are the good algorithms to mine association rules since these algorithms only to generate closed frequent itemsets. Memory consumption of Charm is low when compared to Closet. Both Closet and Charm can find frequent closed pattern when dimensionality is low to moderate. When the number of dimension is very high that is greater than 100, then the efficiency of these algorithms could be significantly impacted.

4 Conclusion
In this paper, we presented and analyzed algorithms such as Apriori, FP-growth, Closet and Charm. Apriori and FP-growth algorithms are used for mining frequent itemsets and Closet, Charm algorithms are used for mining closed frequent itemsets. Except Apriori, FP-growth, Closet and Charm follows depth-first search strategy, their memory requirements are not substantial. Each and every algorithm may run faster than other algorithms in circumstances. Charm can run on very low support values where none of the other methods can be run. Closet & Apriori performs better at high support values.

FP-growth is relatively slow for high minimum support and relatively fast for low minimum support when compared to other methods. Location of data (memory or disk) is non-factor in comparison of algorithms. There is no guarantee through complexity which algorithm to use for efficiency. Based on our analysis, we have observed that all the above discussed algorithms are unique in nature and each & every algorithm performs better in circumstances.

5 References


[11] Jian Pei, Jiawei Han, and Runying Mao, CLOSET: An Efficient Algorithm for Mining Frequent Closed itemsets, Intelligent Database Systems Research Lab, 2000