Market basket Analysis for improving the effectiveness of marketing and sales using Apriori, FP Growth and Eclat Algorithm

Supervisor: Mr. Hossain Arif

Mohammod Akib Khan 13301128

Kazi Mohammad Solaiman 13301038

Touhid Hossain Pritom 13301125

Department of Computer Science and Engineering,

BRAC University

Submitted on: 21st August 2017

DECLARATION

We, hereby declare that this thesis is based on the results found by ourselves. Materials of work found by other researcher are mentioned by reference. This Thesis, neither in whole or in part, has been previously submitted for any degree.

Signature of Supervisor

Mr. Hossain Arif

Signature of Author

Mohammod Akib Khan

Kazi Mohammad Solaiman

Touhid Hossain Pritom

ACKNOWLEDGEMENTS

All thanks to the Almighty **ALLAH(SWT)**, The creator and the owner of this universe, the most merciful, beneficent and the most gracious, who provided us guidance, strength and abilities to complete the thesis.

We are especially thankful to Mr. Hossain Arif sir, our thesis supervisor, for his help, guidance and support in completion of our project. We also are thankful to the BRAC University Faculty Staffs of the Computer Science & Engineering, who have been a light of guidance for us in the whole study period at BRAC University, particularly in building our base in education and enhancing our knowledge.

Finally, we would like to express our sincere gratefulness to our beloved parents, brothers and sisters for their love, care and support. We are grateful to all of our friends who helped us.

Contents

List of figures:

List of Tables:

ABSTRACT

Data mining approach with the help of best frequent pattern extracting algorithm can have a big impact in the field of marketing and sales. Frequent pattern mining is a widely researched field in data mining because of its importance in many real life applications. In this thesis, we used the three most popular algorithms in frequent pattern mining for market basket analysis – FP Growth, Apriori, and Eclat. The design and implementation of these three pattern mining algorithms were discussed in detail. All the three algorithms gave consistent output. We did performance comparison and analysis of these algorithms using three different datasets. Recommendations are provided to suggest the best algorithm to use in different contexts.

1. Introduction:

1.1 Motivation:

In this modern era, marketing and sales has become a big issue in day to day life. Everyday huge number of transaction is happening all over the world. Every transaction and sell pattern is varying from culture to culture and also in different location too. Without the process of data mining it is impossible to predict and analyze a transaction pattern in a specific culture or location. This helps us to think how an organization or an entrepreneur can start a successful business with the help of previous sales data. If we go out for shopping we usually buy products that are being cross matched with each other but we do not realize that. Suppose, in Bangladesh if someone goes to shopping the percentage that person will buy rice and pulse at the same time is very high. But it is being overlooked most of the time. This is just one scenario. Like this there are many pattern of buying a customer regularly follows unconsciously. So if the owner can use this huge repository of past data and analyze them he will get a frequent item that are most likely followed by people of specific location. This idea encourages us the most. With the help of frequent pattern mining algorithm we can come to an inference that the more backward we look the more forward we will see.

1.2 Key Words

- Data mining
- Association Rule Mining
- Market basket
- Apriori, FP Growth, Eclat

2. Literature review:

We read some scientific papers related to our topic and got some informations from those papers. [1] Saurabh Malgaonkar, Sakshi Surve and Tejas Hirave describe a perfect method to extract the data from the huge sets of marketing datasets efficiently by using different techniques of Association Rule Mining(ARM). The systems satisfied the following objectives – to make more informed decision about product placement, pricing, promotion and profitability. Also [2] Mahmoud Houshmand and Mohammad Alishahi from Sharif University of Technology, Iran especially focused on customer behaviour. [3] Another paper written by Xiaohui Yu, Yang Liu, Jimmy Xiangji Huang, Aijun An find out which product should be cross-sold. [1] Saurabh Malgaonkar, Sakshi Surve and Tejas Hirave also describe that their system identifies customer purchasing habits. It provides insight into the combination of products within a customer's baskets. The term 'basket' normally applies to a single order. However, the analysis can be applied to other. We often compare all orders associated with a single customer. Ultimately, the purchasing insights provide the potential to create cross sell propositions. Moreover [4] Julie Marcoux, and Sid-Ahmed Selouani put emphasize on sales forecasting. Sales forecasting is an important part of business management since it provides relevant information that can be used to make strategic business decisions. Forecasting can be divided in three categories: future forecasts, environmental forecasts and industry forecasts. Sales forecasting, considered as a company forecast, aims at assessing the performance of a given company regardless of its competitors. [20] Abbas, Ahmed, Zaini discussed in their paper that one of the problems regarding data mining is to search for meaningful relations in computer purchase data. They also discussed that how differently arranging products in shops decreased the sells of particular items. So they are using market basket analysis to identify purchasing pattern to help retailers to make a better arrangement of the products.

3. System Implementation & Design

Numerous business enterprise store huge amount of data from their everyday operations. For example, at the checkout counter of a super shop a huge number of data is recorded or stored. Table 3.1 shows an example of market basket transaction. Every row of this table represents a transaction which contains a unique TID (Transaction ID) and items. Analyzing this data retailers want to find purchasing behavior of the customers. Such information is helpful for marketing advertising, catalogue management and customer relationship management [21].

Table 3.1. An example of market basket transaction

TID	Items
1	{ Bread, Milk }
2	{ Bread, Diapers, Beer, Egg }
3	{ Milk, Diapers, Beer, Cola }
4	{ Bread, Milk, Diapers, Beer }
5	{ Bread, Milk, Diapers, Cola }

With the help of association analysis we can discover exciting relationship concealed in large datasets. The revealed relationships can be used to represent in the form of **association-rules** or sets of frequent items. For rule can be extracted from the table 3.1:

$$\{\text{Diapers}\} \rightarrow \{\text{Beer}\}$$

The rule recommends there lies a strong sales relationship between diapers and beer because many customer who buy diapers also buy beer. Using this interesting results retailers can arrange their products in a way that increases their sale[21].

There are basically two major issues for applying association analysis to market basket data.

- At first, we need we need to discover patterns from a large transaction dataset.
- Secondly some of the pattern that was found can be spurious because they can occur simply.

At first we describe the fundamental ideas of association analysis and the algorithms that we used to mine the frequent patterns from a large dataset. Last of all we will talk about how we can prevent the results that are spurious [21].

3.1 Problem Definition

Some fundamental terminology that are used in association analysis and patterns a formal explanation of the task is.

Binary Representation We can store market basket data in a binary format shown in table3.2, where each row and column signifies a transaction and an item respectively. An item is considered a binary variable whose value is one when it is present in the transaction and zero if it is not present. Because we are more concerned about the presence of an item rather than it's absence, an item is an asymmetric binary variable [21].

Table 3.2. A binary presentation of market basket data.

TID	Bread	Milk	Diapers	Beer	Eggs	Cola
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	0
5	1	1	1	0	0	1

This illustration is a very simple representation of market basket data because it is not concerned about important features of the data like how many copies was sold and the price of the item.

Itemset and Support Count: We consider $I = \{i_1, i_2, \ldots\ldots\ldots, i_d\}$ as the set of all items in a market basket data and $T = \{t_1, t_2, \ldots\ldots\ldots, t_N\}$ is the set of all transactions. Each transaction t_i has a subset of items chosen from I. In Association analysis, an itemset is a collection of zero or more items. If an itemset contains k items, it is called k-itemset. For example {Diapers, Bread, Milk} is an item of a 3-itemset. An itemset is considered as a null (or empty) set if it has no item in it.

The transaction width is calculated by the number of items present in a transaction. An important stuff of an itemset is **support count,** which refers to the number of transaction that contain a specific itemset. Mathematically, the support count, σ(X), for an itemset X can be started as follows:

$$\sigma(X) = |\{t_i | X \subseteq t_i , t_i \in T\}|$$

In table 1.1 the support count for {Bread, Milk} is equal to three because they occurs in only three transaction.

Association Rules Association rule is an implication expression of form $X \rightarrow Y$, where X and Y are disjoint itemsets, that is $X \cap Y = \emptyset$. The strength of an association rule can be measured in terms of its **support** and **confidence** [21]**.**

- Support determines how frequent a rule is appropriate to a given data set.
- Confidence determines how frequently items in Y appear in transactions that contains X.

The formal definition of these metrics are

$$\text{Support, } s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N} \qquad (3.1)$$

$$\text{Confidence, } c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \qquad (3.2)$$

Example: Let us consider the rule {Milk, Diapers} → {Bear}. Since the support count for {Milk, Diapers, Beer} is 2 and total number of transaction is 5, the rule's support is 0.4.

$$\text{Support, s } (\{\text{Milk, Diapers}\} \rightarrow \{\text{Bear}\}) = \frac{\sigma(Milk,Diapers,Beer)}{N} = \frac{2}{5} = 0.4$$

The rules confidence is gained by dividing the support count for {Milk, Diapers, Beer} by the support count for {Milk, Diapers}. Since there are 3 transactions that contain milk and diapers the confidence is 0.67 [21].

$$\text{Confidence, c } (\{\text{Milk, Diapers}\} \rightarrow \{\text{Bear}\}) = \frac{\sigma(Milk,Diapers,Bear)}{\sigma(Milk,Diapers)} = \frac{2}{3} = 0.67$$

Why Use Support and Confidence? Support is very essential measure as it indicates how frequent any item has purchased. A rule that has low support that may occur by chance. A low support rule is also likely to be uninteresting and irrelevant to business as it is not be profitable to promote items that customers seldom buy together. For these reasons, support is often used to eliminate uninteresting rules. As will be shown in 3.2.1, support also has a desirable property that can be exploited for the efficient discovery of association rules.

Whereas, confidence calculates the dependability of the inference made by a rule. For given X → Y, the higher the confidence, the more likely it is for Y to be present in transaction that contain X. Confidence can also be considered as an estimation of the conditional probability of Y given X [21].

The result of the Association analysis should be inferred with cautiousness. Instead of implying causality, it suggests a strong co-occurrence relationship between items in the antecedent and consequent of the rule. On the other hand, causality needs knowledge and effect attributes in data [21].

Formulation of Association Rule Mining Problem The association rule mine problem can be formally stated as follows:

Definition 3.1 (Association Rule Discovery). If a transaction T is given, we tends to find all the rules of which support ≥ minsup and confidence ≥ minconf, where minsup and min conf are the corresponding support and confidence thresholds [21].

A brute-force approach for mining association rules can be done by computing the support and confidence for every possible rule. As there are exponentially many rules that can be extracted from a data set this approach is excessively expensive. Moreover from a dataset of d items can possible extract the following number of rules,

$$R = 3^d - 2^{d+1} + 1. \qquad (3.3)$$

Even for data set which is small shown in table 4.1, this approach requires us to compute support and confidence for $3^6 - 2^7 + 1 = 602$ rules. As 80% of the rules are castoff if we apply minsup = 20% and minconf = 50%, thus most of the calculation became wasted. If we can prune the rules early without computing their support and confidence, we can reduce the unnecessary computation [21].

By decoupling the support and confidence requirements initially, we can improve the association rule mining algorithms. From Equation 3.2, it can be inferred that the support of a rule $X \rightarrow Y$ is only dependent on the support of its corresponding itemset, $X \cup Y$. For instance, support of the following rules is same because they contain items from the same itemset, {Beer, Diapers, Milk}:

{Beer, Diapers} → {Milk}, {Beer, Milk} → {Diapers},

{Diapers, Milk} → {Bear}, {Beer} → {Diapers, Milk},

{Milk} → {Beer, Diapers}, {Diapers} → {Beer, Milk}.

We can prune all off the six candidate rules immediately without computing their confidence value if the itemset is infrequent.

A common approach used by Association rule mining algorithms can be divided into two major subtasks:

1. **Frequent Itemset Generation:** The objective of this is to find all the itemsets that satisfy minimum threshold. These itemsets are called frequent itemsets.
2. **Rule Generation:** Its objective is to extract all the high-confidence rules from the frequent itemsets found in the previous step. These rules are called strong rules.

Frequent itemset generation is computationally expensive than those of rule generation. Efficient techniques for generating frequent itemsets and association rules are discussed in Section 3.2 and 3.3 respectively [21].

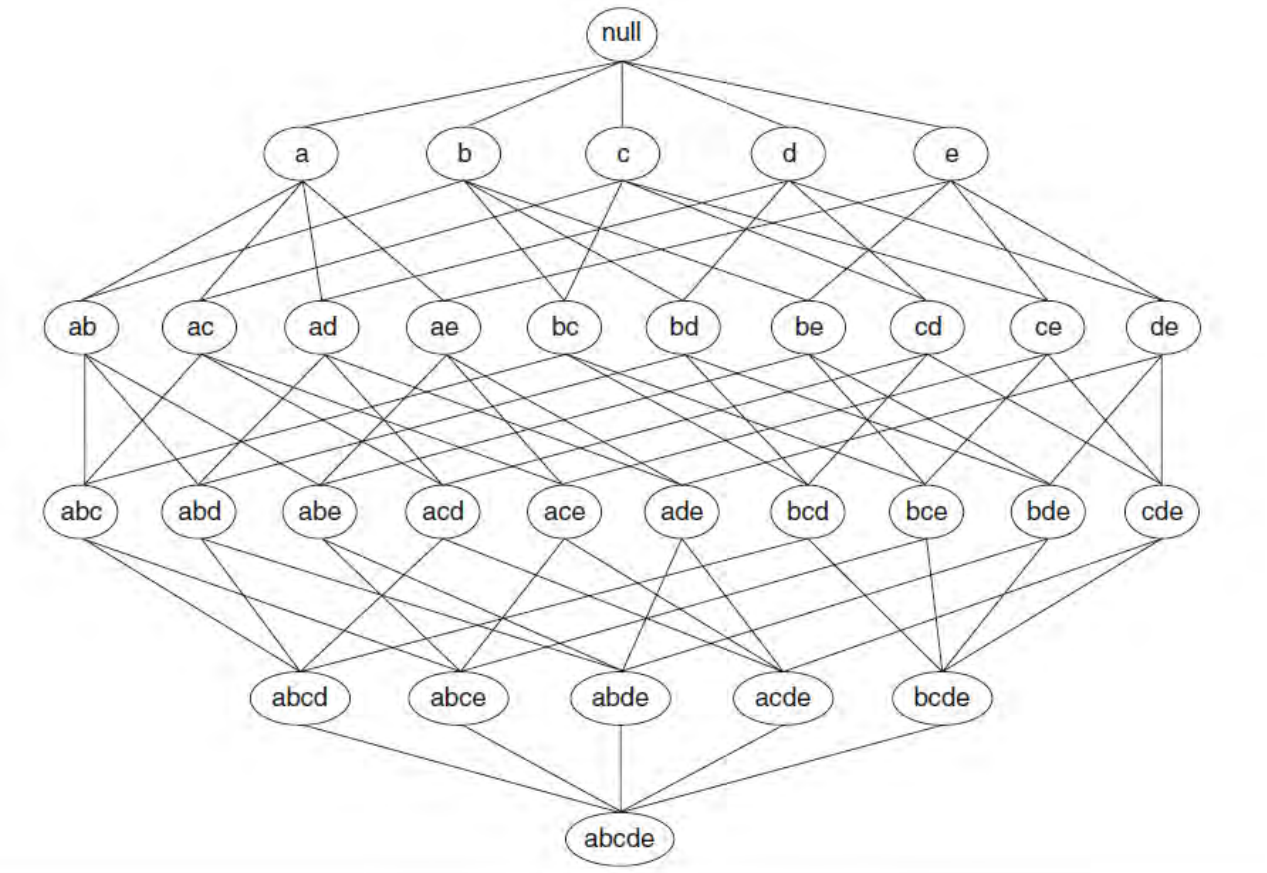

Figure 3.1. An Itemset lattice

3.2 Frequent Itemset Generation

A lattice structure can be used to compute the list of all possible item sets. Figure 4.1 shows an itemset lattice for I = {a, b, c, d, c, e}. A data set that that has k items can possibly generate up to $2^k - 1$ frequent item sets, excluding the null set. Usually k is very large in many practical applications, therefor the search space of item sets can be exponentially large [21].

A brute-force approach for finding frequent itemsets is to determine in the support count for every **candidate itemset** in the lattice structure. We can complete this task by comparing each candidate against every transaction, a process that is shown in Figure 3.2. The occurrence of a candidate in a transaction will increment the support count. For example, the support for {Bread, Milk} is incremented to three as the itemset has occurred in transactions 1, 4, and 5. Such an approach can be very expensive because it requires o (N M w) comparisons where,

- N is the number of transactions
- M = 2^k -1is the number of candidate itemsets, and
- w is the maximum transaction width [21].

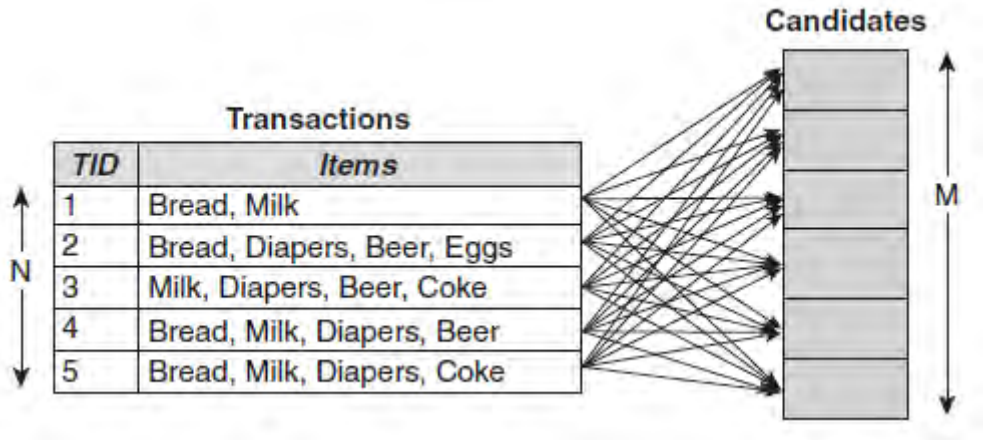

Figure 3.2. Counting the support of candidate itemsets.

Computational complexity of frequent itemset generation can be reduced by the following mentioned task.

1. **Reduce the number of candidate itemsets (M):** With the help of Apriori principle described in the next section, without counting the support values of candidate itemsets we can eliminate some of them.
2. **Reduce the number of comparisons:** We can decrease the number of comparisons with the help of more advanced data structures, either to store the candidate itemsets or to compress the data set instead of matching each candidate with each transaction. We will discuss these strategies in later sections [21].

3.2.1 The Apriori Principle

Apriori principle reduces the number of candidate itemsets explored while generating frequent itemsets with the help of support measure. Pruning is done with the help of support which is guided by the following principle.

Theorem (Apriori Principle): If an itemset is frequent, then all of its subsets must also be frequent [21].

For understanding the Apriori principle, let us consider the itemset lattice shown in Figure 4.3. if {c, d, e} is a frequent itemset, then all subsets of {c, d}, {c, e}, {d, e}, {c}, {d} and {e}. As a result, if {c, d, e} is frequent, then all subsets of {c, d, e} must also be frequent.

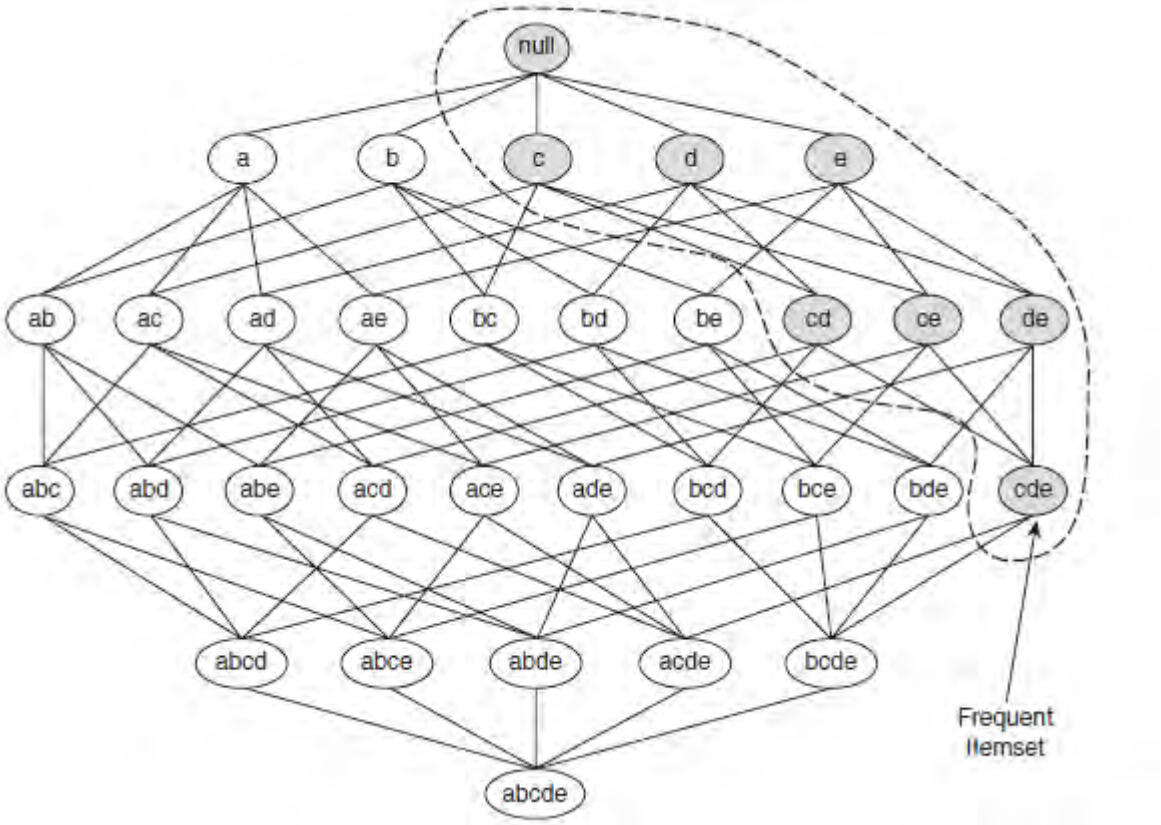

Figure 3.3. An Illustration of Apriori that shows if {c, d, e} is frequent

Then all of its subset is also frequent

On the contrary, if any itemset such as {a, b} is infrequent then all of its supersets must be infrequent too. As shown in figure 3.4, the entire subgraph containing the supersets of {a, b} can be pruned immediately once {a, b} is infrequent. This pruning technique of the exponential search space based on the support measure known as **support-based pruning**. Such a pruning strategy is made possible by a key property of the support measure, namely, that the support of an itemset can never exceed the support for its subsets. This property is also known as the **anti-monotone** property of the support measure [21].

Definition (Monotonicity Property): Let I be a set of items, and $J = 2^I$ be the power set of I. A measure f is monotone (or upward closed) if

$$\forall \boldsymbol{X}, \boldsymbol{Y} \in \boldsymbol{j} : (\boldsymbol{X} \subseteq \boldsymbol{Y}) \rightarrow \boldsymbol{f}(\boldsymbol{X}) \leq \boldsymbol{f}(\boldsymbol{Y}),$$

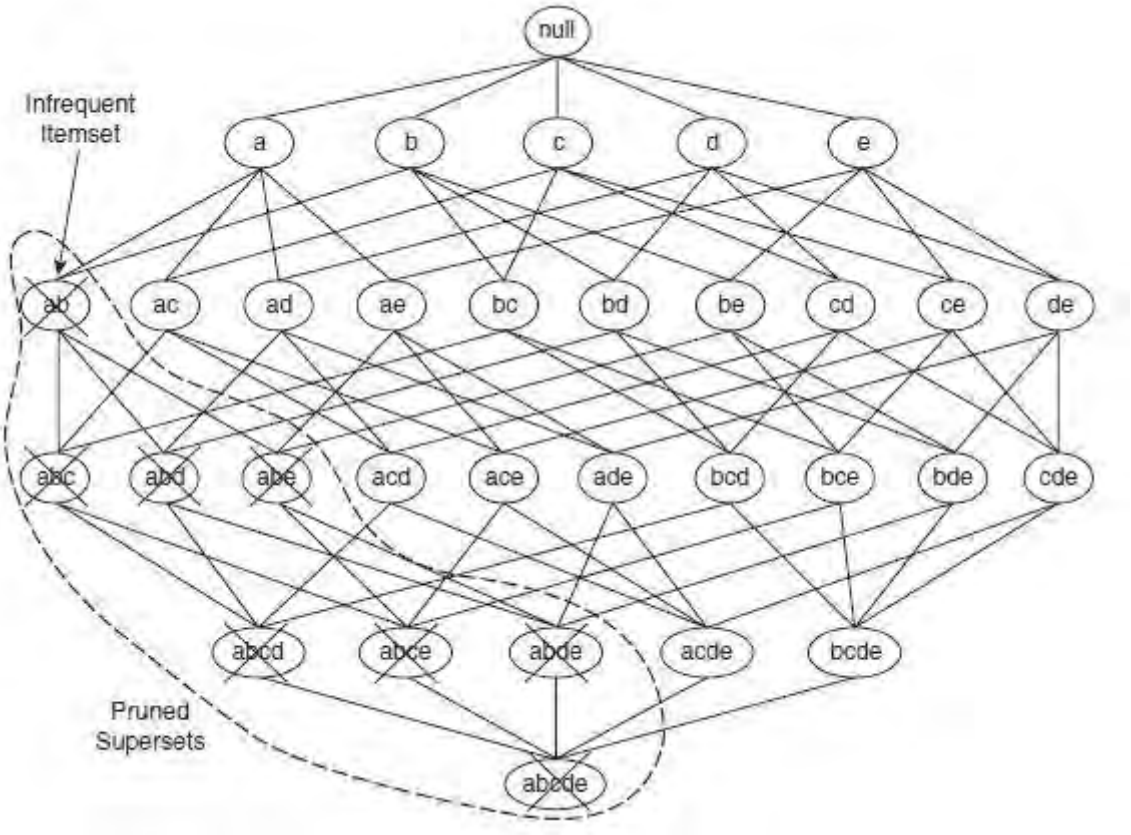

Figure 3.4 An illustration of support-based pruning. If {a, b} is infrequent, then all supersets of {a, b} are infrequent

which means if X is a subset of Y, then f(Y) must not exceed f(X) [21].

In order to successfully prune the exponential growth of candidate itemsets, any measures that possesses an anti-monotone property can be amalgamated directly into the mining algorithms, as will be shown in the next section [21].

3.2.1.1 Frequent Itemset Generation in the Apriori Algorithm

Apriori is the first association rule mining algorithm that pioneered the use of support-based pruning to systematically control the exponential growth of candidate itemsets. Figure 4.5 offers

a high-level illustrations of the frequent itemset generation part of the Apriori algorithm for the transaction shown in Table6.1. We assume that the support of threshold is 60%, which is equivalent to a minimum support count equal to 3 [21].

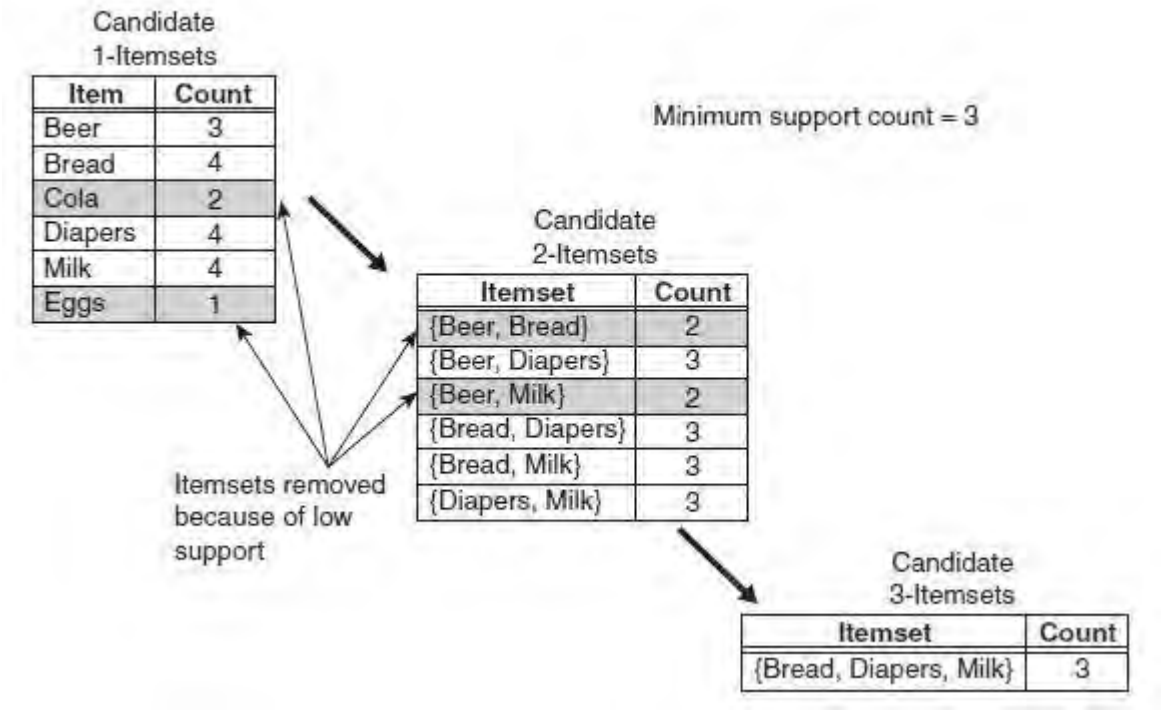

Figure 3.5 Illustration of frequent itemset generation using Apriori Algorithm

Initially, every item is considered as a candidate 1-itemset. After that we count their supports and discarded the itemset which appear fewer than three transaction. As a result {Cola} and {Eggs} are discarded. In the following iteration, with the help of frequent 1-itenset candidate 2-itemset is generated because the Apriori principle guarantees that all supersets of the infrequent 1-itemsets must be infrequent. Because there are only four frequent 1-itemset, the number of candidate 2-itemsets are generated by the algorithm is $({}^{4}C_{2}) = 6$.

The efficiency of the pruning strategy can be shown by counting the number of candidate itemsets generated. A brute-force strategy of enumerating al itemsets (up to size 3) as candidates will produce 41 candidates [21].

$$({}^{6}C_{1}) + ({}^{6}C_{2}) + ({}^{6}C_{3}) = 6 + 15 + 20 = 41$$

Where the Apriori principle, this number decrease to 13.

$$({}^{6}C_1) + ({}^{4}C_2) + 1 = 6 + 6 + 1 = 13$$

This shows a 68% reduction in the number of candidate itemsets even in a simple example [21].

The pseudo code for the frequent itemset generation part of the apriori algorithm shown in Algorithm 3.1 Let C_k denote the set of candidate k-itemsets and F_k denote the set of frequent k-itemsets:

Algorithm 3.1 Frequent itemset generation of the Apriori algorithm.

1. K = 1.
2. $F_k = \{i \mid i \in I \wedge \sigma(\{i\}) \geq N \times minsup\}$. {Find all frequent 1-itemsets}
3. **repeat**
4. k = k+1.
5. C_k = apriori –gen (F_{k-1}).
6. **for** each transaction t ∈ T **do**
7. C_t = subset(C_k, t).
8. **for** each transaction candidate itemsets c ∈ T **do**
9. $\sigma(c) = \sigma(c) + 1$
10. **end for**
11. **end for**
12. $F_k = \{ c \mid c \in C_k \wedge \sigma(c) \geq N \times minsup \}$
13. **until** $F_k = \emptyset$.
14. Result = $\cup F_k$

3.2.1.2 Candidate Generation and Pruning

Step 5 of Algorithm 3.1 the Apriori-gen function generates candidate itemsets by performing the following operations:

1. **Candidate Generation:** This operation generates new candidate k-itemsets based on the frequent (K -1) itemsets found in the previous iteration.

2. **Candidate Pruning:** This operation eliminates some of the candidate k-itemsets using the support-based pruning strategy [21].

3.2.1.3 Support Counting Using a hash Tree

In the Apriori algorithm, candidate itemsets are separated into different buckets and stored in a hash tree. While counting the support of an itemset, itemsets contained in each transaction are hashed into their appropriate buckets. This helps to hash every transaction into the proper buckets. This reduces the comparison of each itemset in the transaction with every candidate itemsets, it is matched only against candidate itemsets that belong to the bucket [21], as shown in Figure 3.6

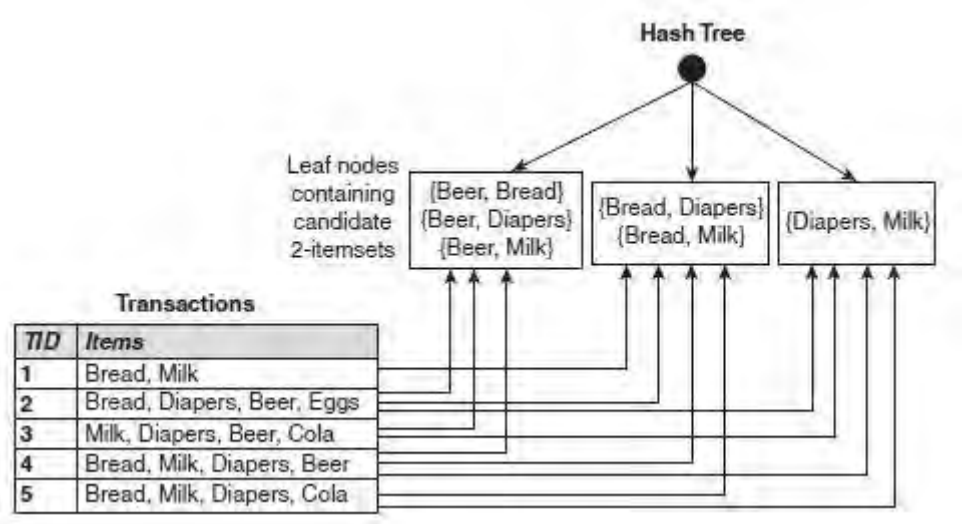

Figure 3.6 Counting the support of itemsets using hash tree.

Figure 3.7 shows an example of a hash tree structure. Each internal node of tree uses the following hash function, h(p) = p mod 3, to determine which branch of the current node should be followed next. For example 1, 4, 7 are hashed to the same branch (i.e., the left most branch) because they have the same remainder after dividing the number by 3. All candidate itemsets stored at the leaf nodes of the hash tree [21].

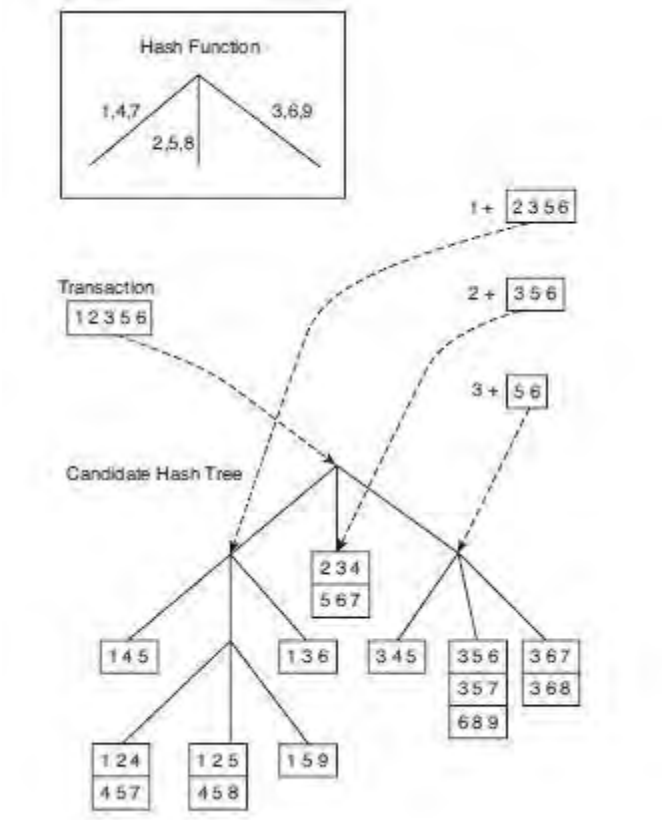

Figure 3.7 Hashing a transaction at root node of a hash tree.

3.2.2 FP-Growth Algorithm

FP-growth is another algorithm to generate frequent itemsets. Fp-growth uses a tree structure called FP-tree and extracts frequent itemsets from the tree structure.

3.2.2.1 FP-Tree representation

An FP-tree is a compresses the input data. To construct the FP-tree one transaction is read from the input data set and mapped to the path of the tree. Common items of different transaction have overlapping path in the FP-tree, this is how compression of data works. So the more overlapping paths the tree has the less memory is needed to store the input data. If the tree size is small enough to for the main to allocate space then does not need to access the hard dish repeatedly frequent pattern generation [21].

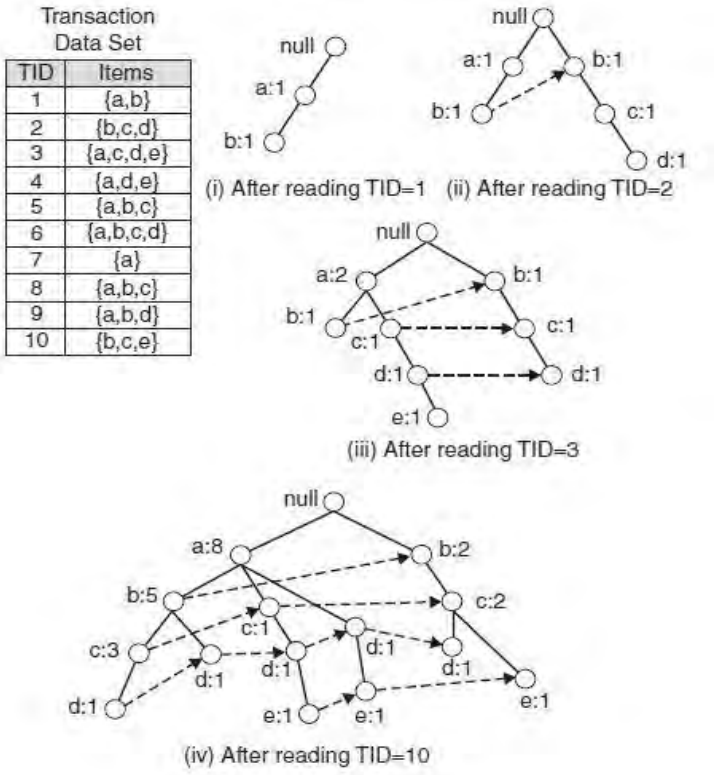

Figure 3.8 Construction of an FP-Tree

The diagram shows a data set that contains transactions and five items. The structures of the FP-tree after taking the first three transactions is also shown in the diagram above. Every node in the tree contains the label of an item and also a tracker which represents the number of transactions that took the given path. The ways in which the FP-tree is generated is shown below:

1. At first the data is scanned to generate the support value of each item. The items which are not frequent are avoided, on the other hand items are sorted in decreasing order which are frequent. The above figure shows that *a* is the most frequent item, after that *b*, then *c*, then *d* and finally *e* [21].
2. Then the algorithm traverse the data again for the construction of the FP-tree. After reading the first transaction {a, b}, the nodes *a* and *b* are created. A path is then formed from root-> a -> b to represent the transaction in the tree. Now the count value of each node is one [21].
3. After that when the second transaction is traversed {b, c, d}, new nodes are created to represents b, c, and d. A path (root->b->c->d) is then formed by connecting the nodes *b*, *c* and *d*. Even though the initial two transaction contain b as common product but their route do not overlap because they have different predecessor [21].
4. Then the third transaction {a,c,d,e}, which have a common predecessor that is *a* with the transaction at the beginning. As long as the predecessor of the item matches they keep overlapping the path [21].
5. Similar process continues until all the data is inserted in the FP-tree [21].

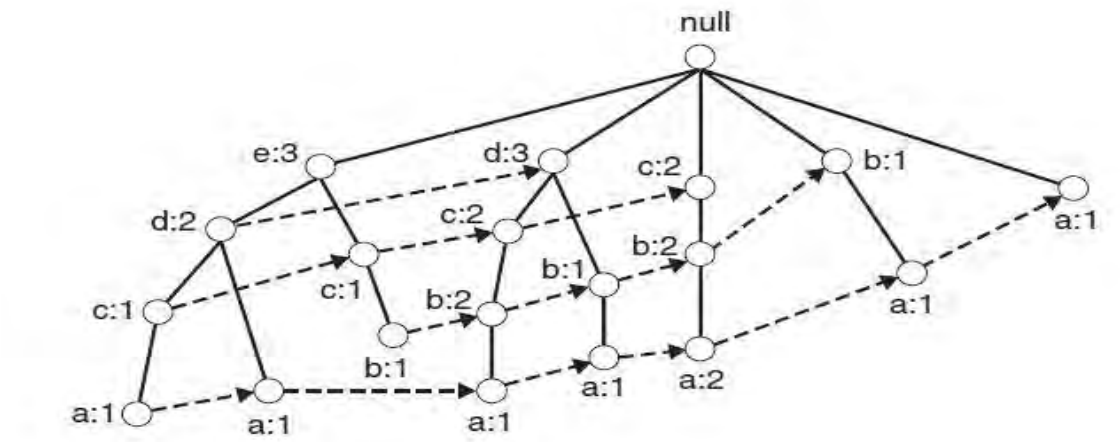

Figure 3.9 An Fp-tree representation for the set shown in fig 3.9 with a different it ordering scheme

3.2.2.2 Frequent itemset Generation in FP-Growth Algorithm

FP-growth is an algorithm that generates frequent itemset from an FP-tree by traversing the FP-tree in a bottom-up approach. In Figure 3.10, the algorithm looks for frequent itemsets that ends with e first, after that by d, c, b and finally a. As all transactions are inserted onto a path in the FP-tree, so we can simply derive the frequent itemsets ending with a certain product by examining only the paths containing that node. The paths are shown in the figure below [21].

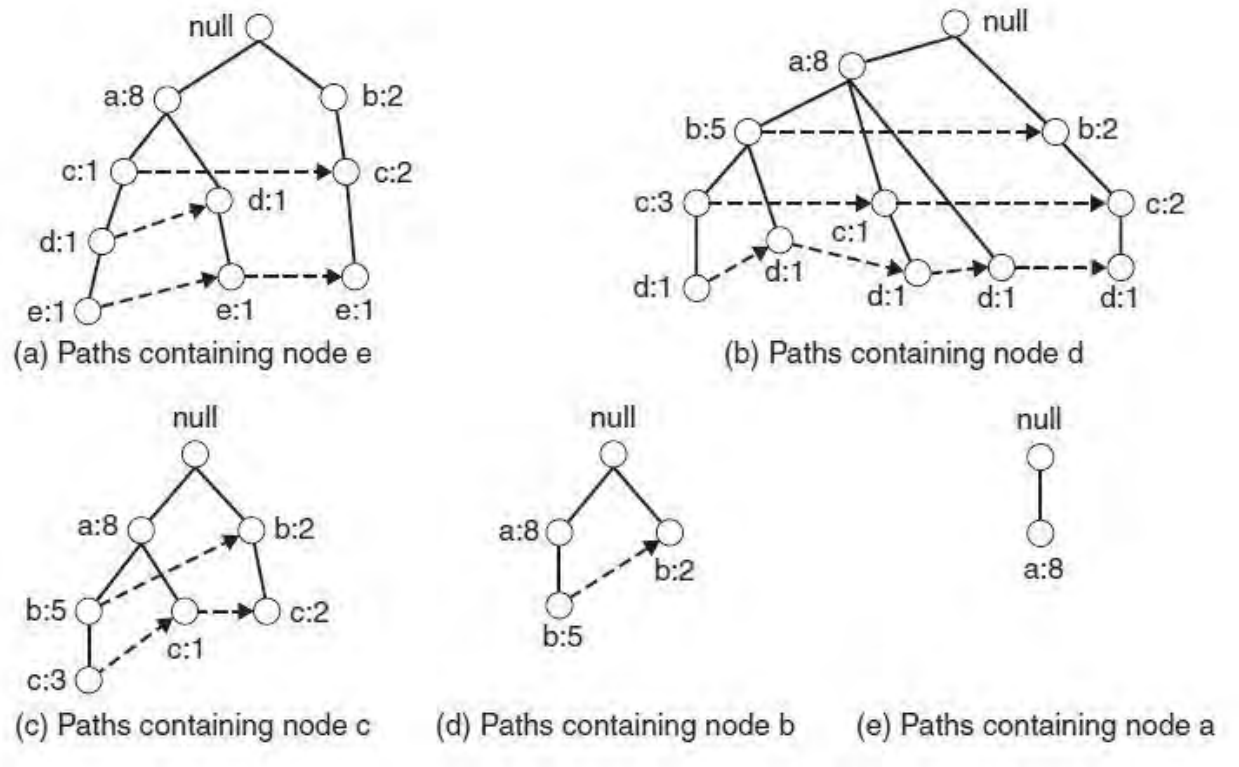

Figure 3.10 Decomposing the frequent itemset generation problem into multiple sub-problems, where each sub problem involves finding frequent itemset ending in e, d, c, b and a.

Table 3.3 The list of frequent itemsets ordered by their corresponding suffixes

Suffix	Frequent itemsets
E	{e}, {d,e}, {a,d,e}, {c,e}, {a,e}
D	{d},{c,d},{b,c,d}, {a,c,d}, {b,d},{a,b,d},{a,d}
C	{c}, {b,c},{a,b,c},{a,c}
B	{b}, {a,b}
A	{a}

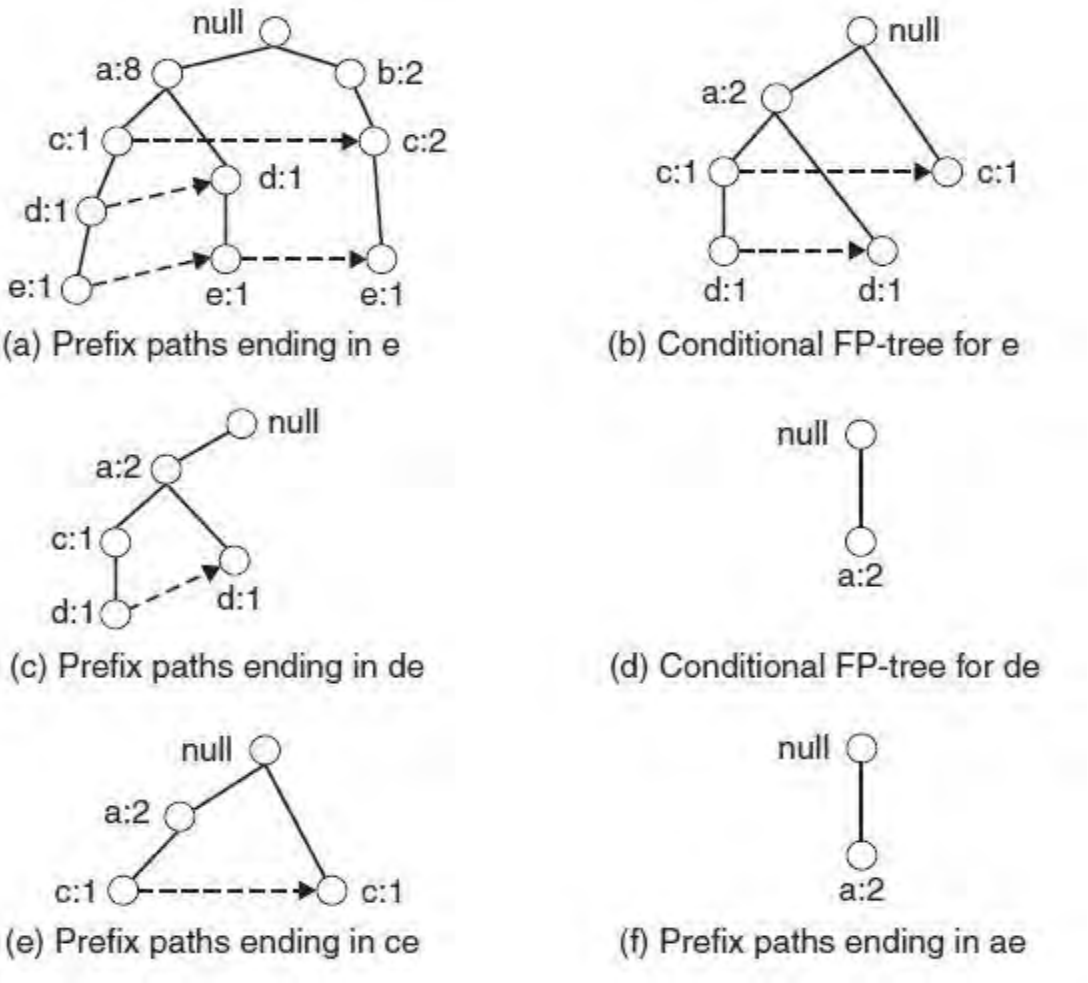

Figure 3.11 Example of applying the FP-growth algorithm to find frequent itemsets ending in e

For a more solid example about solving sub-problems, we are taking the case about finding itemset that ends with *e*.

1. The initial step is to figure out all the route that include node e. These initial paths are known as prefix path and are shown in the above figure.
2. The support value for *e* is calculated by adding the support values associated with node e. Lets say the support value of *e* is 5 and the minimum support value is 4 then *e* is frequent.
3. As *e* is frequent now the algorithm has to solve the sub-problems of finding frequent itemsets ending in de, ce, be and ae. To solving these sub-problems, the algorithm need to

generate conditional FP-tree which is needed to find itemset ending with particular combination of items [21].

3.2.3 Eclat (Equivalence Class Clustering and Bottom Up Lattice Traversal)

Eclat is a vertical database layout algorithm used for mining frequent itemsets. It is based on depth first search algorithm. In the first step the data is represented in a bit matrix form. If the item is bought in a particular transaction the bit is set to one else to zero. After that a prefix tree needs to be constructed. To find the first item for the prefix tree the algorithm uses the intersection of the first row with all other rows, and to create the second child the intersection of the second row is taken with the rows following it [**10**]. Other items of the tree is found in the similar way. The rows which are not frequent are not taken in calculating frequent pattern. Once the tree is constructed eclat then run depth first search algorithms to generate frequent itemsets. The frequent pattern generated through DFS are then stored in a bit matrix. Eclat consumes less memory as is uses prefix tree.

Table 3.4 Horizontal data layout

TID	Items
1	A,B,E
2	B,C,D
3	C,E
4	A,C,D
5	A,B,C,D
6	A,E
7	A,B
8	A,B,C
9	A,C,D
10	B

In the table 3.1 TID represents the transaction id and the alphabets in the capital letters represents the individual product name that are being bought by the customers.

Table 3.5 Vertical data layout

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

In this table 3.5 highlighted letters are representing unique individual products and the numbers are showing the transaction ids of those products that are purchased by the customers.

3.2.3.1 Implementation

In the implementation we have used eclat algorithm along with a virtual transaction data of a super market called chess dataset. The input file contains around three thousand transaction and at first we arrange these data in a bit matrix and then run the eclat algorithm on it.

A storage which contains data attempts to perform tracking using its existing architecture will most noticeably experience significant performance problem. Those performance problems may affect the data storage program to decrease the tracking system a bad fit for the existing program and therefore causing to abandon the system.

We have performed the database connectivity with the input file source (chess.dat, chainstoreFIM.dat, contextPasquier99.dat, foodmartFIM.dat, retail.dat, test.dat) from which we have extracted transactional frequent pattern. After that we can mention the specific conditions as per demand and check the transaction and figure out the best combination. There might be lots of transactions which contain the same itemset.

On the other hand, itemset of many transaction might look different initially but after discarding the infrequent items those itemsets could be identical. Moreover, if there are transactions that do not contain any similar set of items still may have subset which are similar. Such transaction are grouped in the transaction tree. Once the full prefix tree is constructed it contains lots of identical sub trees and thus building a complete prefix tree is very expensive (memory) so the tree is reduced in size by sorting the information of the identical sub trees which helps to build an optimized tree.

3.2.3.2 Implementation Procedure

The procedure for the implementations are:

i. We used vertical tid-list dataset format where we joined with each itemset a list of transactions in which it occurs.[1]
ii. We show that all frequent itemset can be written sequentially in a text file via simple tid-list intersection. [1]
iii. ITable construction which contain all individually frequent items and the support of each item[1]

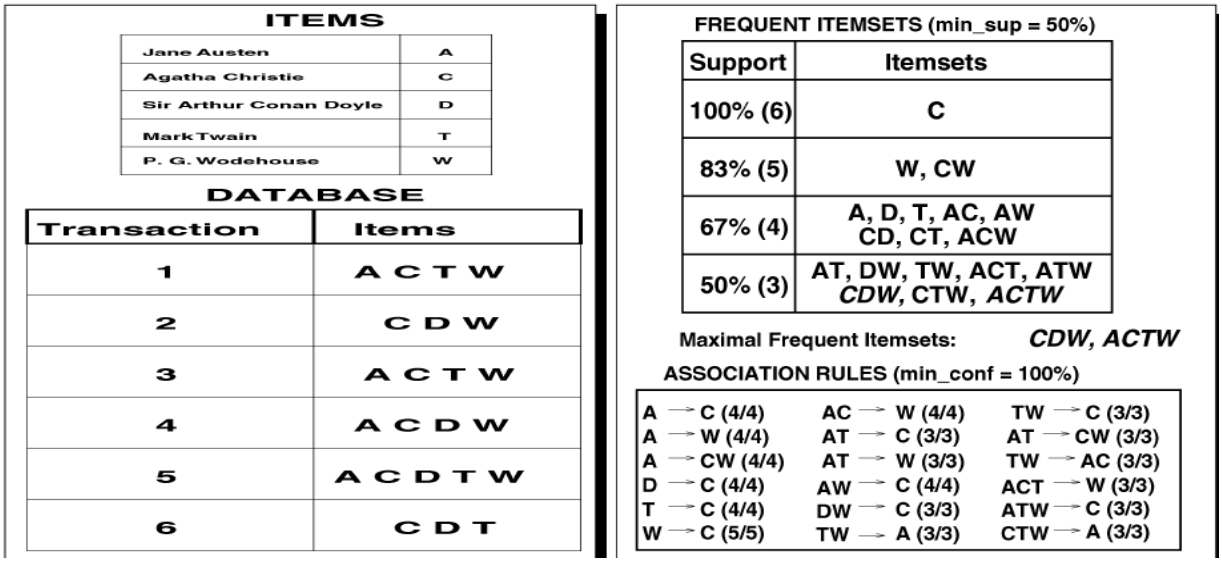

Figure 3.12 Frequent itemsets in eclat

iv. TLink construction which indicates all the transaction of the database containing the frequent items. [1]

Figure 3.13 Basic processing cycle[1]

v. Entries in ITable and Tlink are added or modified by scanning the database.[1]

vi. Frequent items are figured out in the ITable with the help of of their support counts and the infrequent items are discarded from the TLink.[1]

vii. The unnecessary transaction containing set of items whose support fall under the minimum support level are discarded at the beginning [1].

viii. At this stage all the data is trimmed as required and only now the required mining process starts and all the frequent itemset of two and more items are obtained [1].

ix. The database is traversed to search all 1-frequent items and stored in ITable. All contains in the ITable are sorted in the frequency of their ascending order and then the items are mapped to new identifiers that are the increasing sequence of integer values [1].

x. Using the results of above step, only one-frequent items are obtained from the database. They are then represented to the new item identifiers and the transactions are put into the reduced transaction tree. A pointer is maintained to the subset of each item set. This pointer helps to indicate the starting point to mine all frequent item sets corresponding to the paths ending at nodes of the corresponding subset [1].

xi. All the frequent item sets of two or more items are extracted in a recursive fashion [1].

3.2.3.3 Actual mining flowchart:

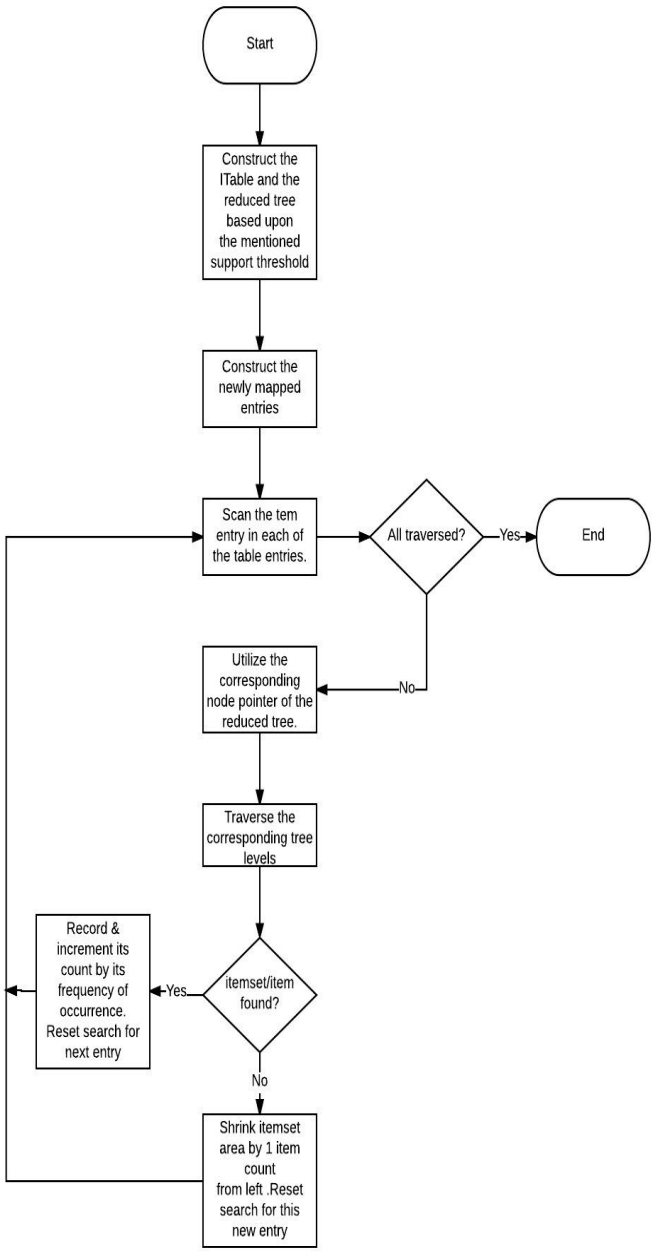

Figure 3.14 Mining workflow

4.Results and Data Analysis:

Frequent pattern mining is the widely research field in data mining because of its importance in many real life applications. We used two most popular algorithms in the frequent pattern mining in our thesis for market basket analysis- FP Growth and Apriori along with Eclat. They all gave efficient output. Still we came across to know the performance analysis of these algorithms in different datasets. Our primary dataset was the accumulation of transactions of a supermarket of different item sets

Apriori, FP Growth and Eclat – all these algorithms now used for frequent pattern mining but there lies some differences among them. Numerous techniques have been experimented for mining frequent pattern rules in the field of research studies. In the arena of frequent pattern rule mining Apriori algorithm is most extensively used that generates frequent patterns[5]. It mines frequent patterns by multiple scans of the database.

Another achievement in the field of frequent pattern mining is the FP Growth algorithm. Han et al, introduced this algorithm which constructs a frequent pattern tree called FP tree. It overcomes two flaws of Apriori algorithm [4][5][7][8].

4.1 Analysis of Apriori:

Now in phase of comparisons first we will talk about Apriori algorithm first in terms of its performance analysis. Apriori is the most classical and and important algorithm for mining item sets which are very frequent. It is initially introduced by Agrawal and Srikant. The main idea of Apriori algorithm is to make multiple passes over the datasets or database in which the transactions or data are saved.

Apriori algorithm depend on Apriori property which states that – “All non empty item sets of a frequent item set must be frequent [8]. It also described a property which describes if the system cannot pass the minimum support test all of its supersets will fail to pass the test. Apriori algorithm uses breadth first search (BFS).

It also uses downward closure property (any super set of an infrequent item set is infrequent that is pruned). It usually adopts horizontal layout to represent the transaction database. The frequency of the item set is computed by counting its occurrence in each transaction.

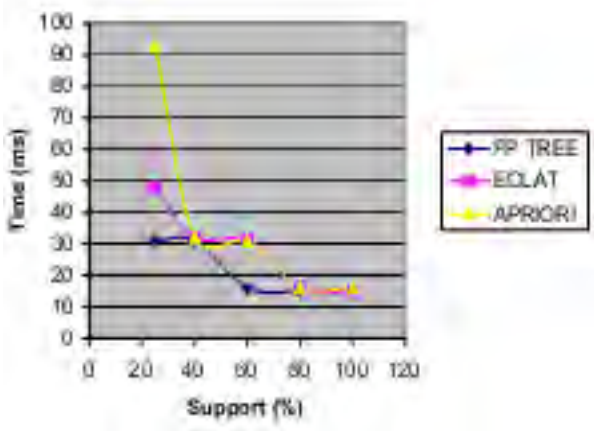

Figure 4.1: Comparison of Apriori, FP Growth and Eclat Algorithm on datasets[10]

Here in this figure, time vs support graph is plotted for Eclat, Apriori and FP Growth algorithm. It is evident from graph that Apriori is taking more time to execute rather FP Growth and Eclat. Eclat takes less time in comparison with those two algorithm.

4.2 Result analysis of Apriori:

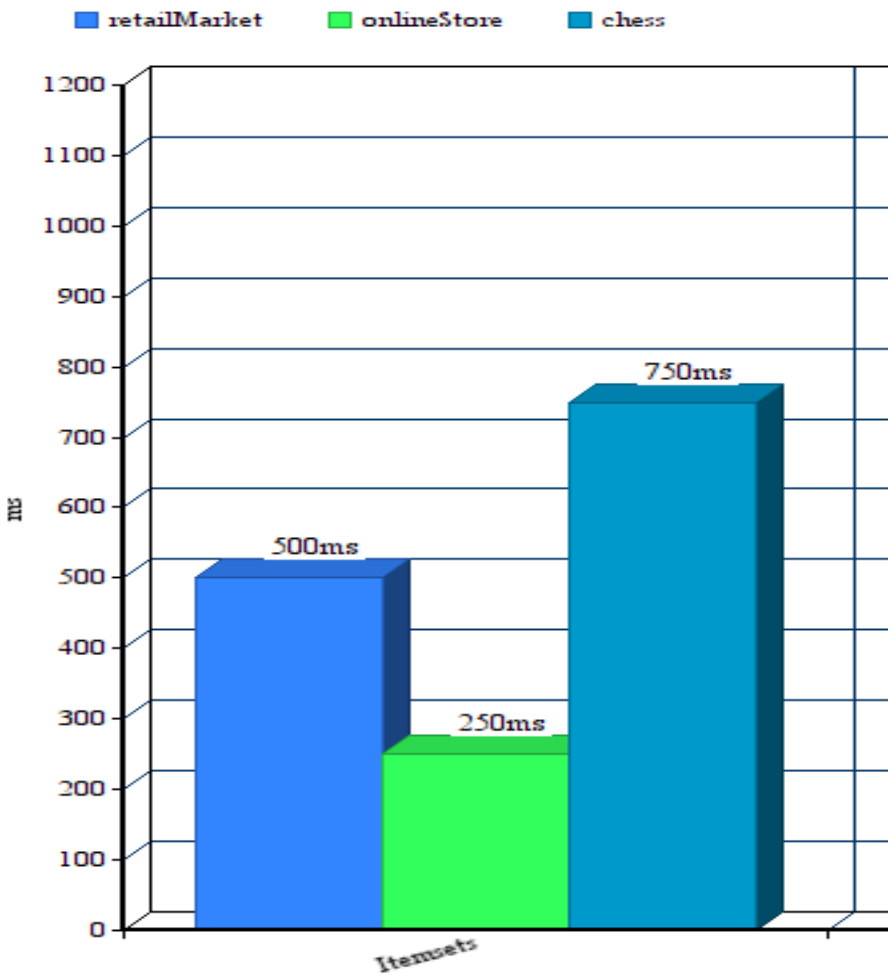

Figure 4.2.1: Performance of Apriori algorithm for different dataset

The figure given above is about analysis of performance of Apriori algorithm in different datasets. Chess.dat was our main datasets. To analyze that dataset and to result a frequent pattern from it Apriori took 350 mille seconds. Chess.dat has over three thousand recorded transaction and each transaction comprised with seventy five different items in each row. For

our thesis we use minimum support of 0.7(means a minimum support of two transaction). It stores data in arrays.

For onlineStore it takes the lowest time as well as used least memory too (4.449 MB) and counted forty-eight thousands seven hundred and thirty-one frequent item sets.

```
=========APRIORI STAT=========
[29, 34, 36, 40, 48, 52, 58, 60, 62, 66]  (0.8031914893617021 2567)
[7, 29, 36, 40, 48, 52, 58, 60, 62, 66]  (0.8050688360450563 2573)
[7, 29, 36, 40, 48, 52, 56, 58, 60, 62]  (0.8016270337922403 2562)
[7, 29, 34, 36, 40, 48, 52, 58, 60, 66]  (0.8041301627033792 2570)
=========APRIORI STAT=========
BUILD SUCCESSFUL (total time: 5 seconds)
```

Figure 4.2.2: Runtime of Apriori algorithm

```
Created 4445 unique itemsets of size 7
Passing through the data to compute the frequency of 4445 itemsets of size 7
[7, 29, 52, 56, 58, 62, 66]  (0.8288485607008761 2649)
[5, 29, 36, 48, 52, 58, 60]  (0.8504380475594493 2718)
[34, 36, 40, 48, 52, 58, 66]  (0.8388610763454318 2681)
[29, 36, 40, 48, 60, 62, 66]  (0.8432415519399249 2695)
[5, 29, 40, 56, 58, 60, 62]  (0.8319774718397998 2659)
[5, 36, 48, 52, 58, 60, 62]  (0.817584480600751 2613)
[7, 29, 34, 36, 40, 60, 66]  (0.831351689612015 2657)
[34, 40, 48, 56, 58, 60, 66]  (0.8010012515644556 2560)
[29, 34, 36, 40, 58, 62, 66]  (0.8301001251564456 2653)
[5, 29, 36, 48, 52, 58, 62]  (0.8235294117647058 2632)
[7, 29, 36, 56, 58, 60, 62]  (0.8363579474342928 2673)
[3, 29, 40, 48, 52, 58, 60]  (0.818523153942428 2616)
[5, 34, 36, 52, 56, 58, 60]  (0.8097622027534418 2588)
[29, 34, 36, 52, 58, 62, 66]  (0.8310387984981227 2656)
[3, 29, 36, 40, 52, 58, 62]  (0.8157071339173968 2607)
```

Figure 4.2.3: Unique item set with their corresponding support count in Apriori

4.3 Analysis of FP Growth Algorithm:

In terms of FP Growth (Frequent Pattern Growth) this algorithm uses divide and conquer strategy and uses FP data structure to to achieve a condensed representation of transactional database. The idea was given by (han et al 2000) [comparing]. It needs no candidate frequent item sets. Rather than frequent patterns are mined from the fp tree.

In the initial step of fp growth, a list of regular item set is created and arranged in their decreasing support order. This list is represented by a structure called node. Every node in the fp tree, other than the root node, will contain the item name, support count, and a pointer to connect to a node in the tree that has a similar item name [6]. These nodes are utilized to make the fp tree.

Common prefixes can be shared amid FP tree development. The ways from root to leaf node are organized in non increasing order of their support. Once the fp tree is built, at this point frequent patterns are extracted from the FP tree beginning from the leaf nodes. FP Growth takes least memory because of projected layout and is storage efficient [1][2][3]. It has two major steps.

- Contrast a compact data structure called fp tree
- Discover frequent items directly from fp tree.

4.4 Result of FP Growth algorithm

We have performed FP Growth for different datasets. But the initial dataset was chess.dat. in this dataset the transaction count is three thousand one hundred and ninety-six. Maximum memory usage was 3.929 MB. Frequent item set generation count was forty-eight thousand seven hundred and thirty-one. Both the algorithm has given the same result yet in comparison fp growth is faster in performance. It outperforms Apriori both in memory and in time constraints.

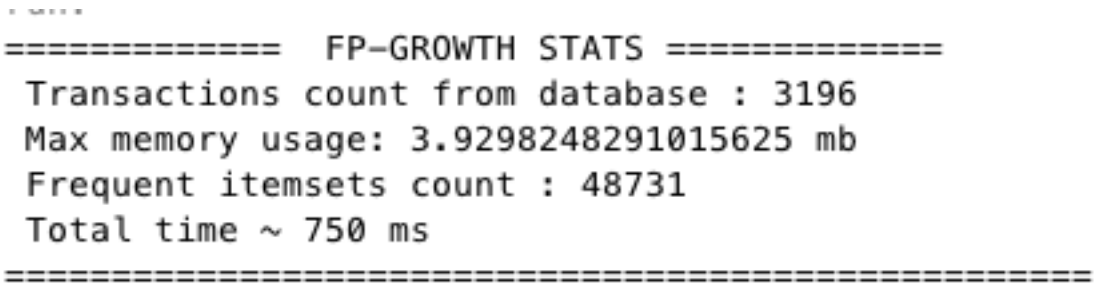

Figure 4.4.1: Performance statistics of FP Growth in chess.dat dataset

In another dataset foodmartFim.dat fp counts four thousand and forty-one transactions from the database. It takes about 6.5 MB of memory and total time of one twenty-seven mille second. But interestingly there are no frequent item set generated in this huge transactional database.

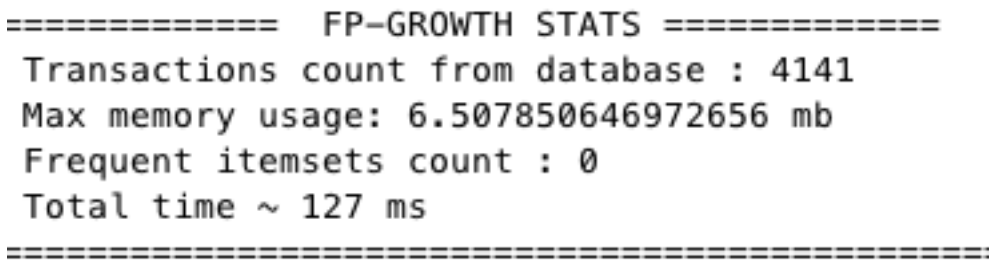

Figure 4.4.2: FP Growth statistics for foodmartFim database

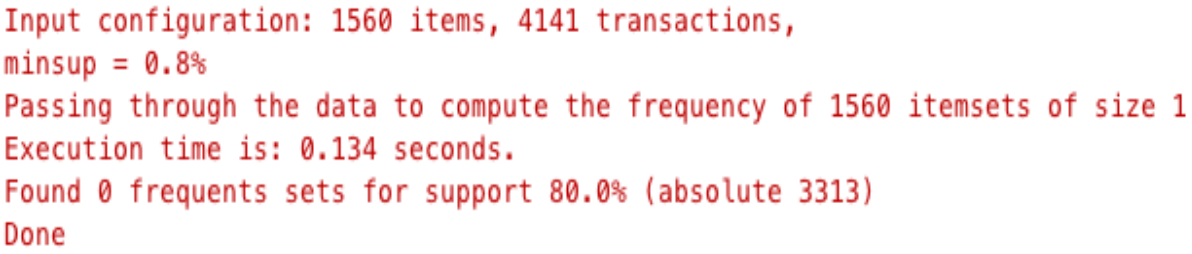

Figure 4.4.3: Input configuration and minsup statistics of FP growth

In this set of data, the transaction counted is over four thousand among fifteen hundred sixty items. The minimum support count is 80%. Still in this set there are no frequent item set is generated. Whereas, in another dataset there were less items and less transactions were recorded but item set that Apriori and FP Growth had found is huge and it generates the number ranging from one to 10 frequent item set generation that are being bought by the customer.Different dataset has different input and not all stores have the same amount of transaction that's why it differs from store to store.

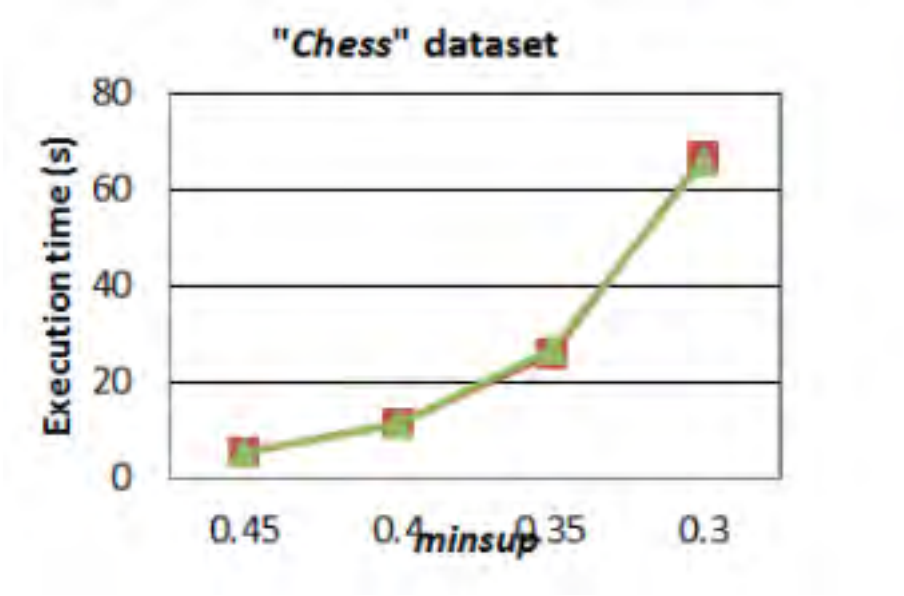

Figure 4.4.4: Performance comparison of FP Growth implementation in chess dataset

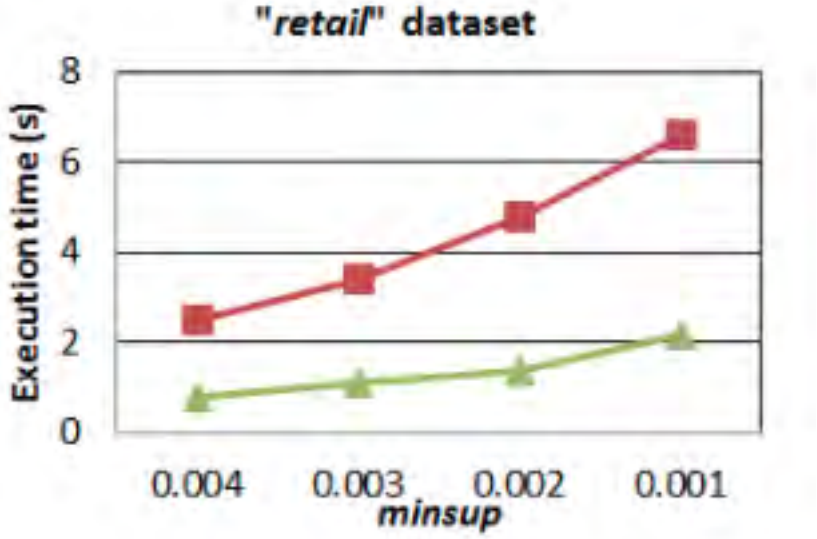

Figure 4.4.5: Performance comparison of FP Growth implementation in retail dataset

```
=============  FP-GROWTH STATS in RETAIL DATASET =======
 Transactions count from database : 88162
 Max memory usage: 57.678741455078125 mb
 Frequent itemsets count : 0
 Total time ~ 620 ms
================================================
```

Figure 4.4.6: retail.dataset result in FP Growth

The figure 4.4.6 shows the result of FP Growth in the retail dataset where total transaction is eight-eight thousand one hundred and sixty-two. To calculate among these datasets FP Growth took 620 mille second.

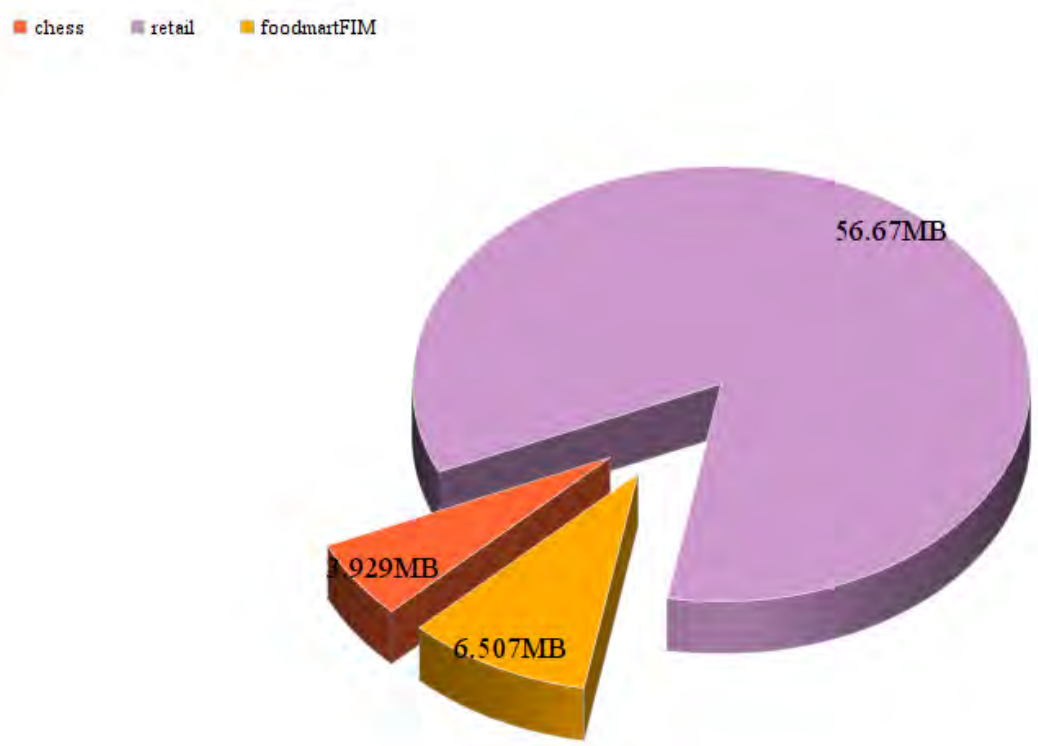

Figure 4.4.7: Memory usage of FP Growth Algorithm in different datasets

Fp growth takes more time in retail and foodmartFIM dataset that it takes in chess dataset. However, only chess dataset generates frequent item sets and other two do not. But when it comes to run time than chess dataset takes much time than retail and foodmartFIM datasets.

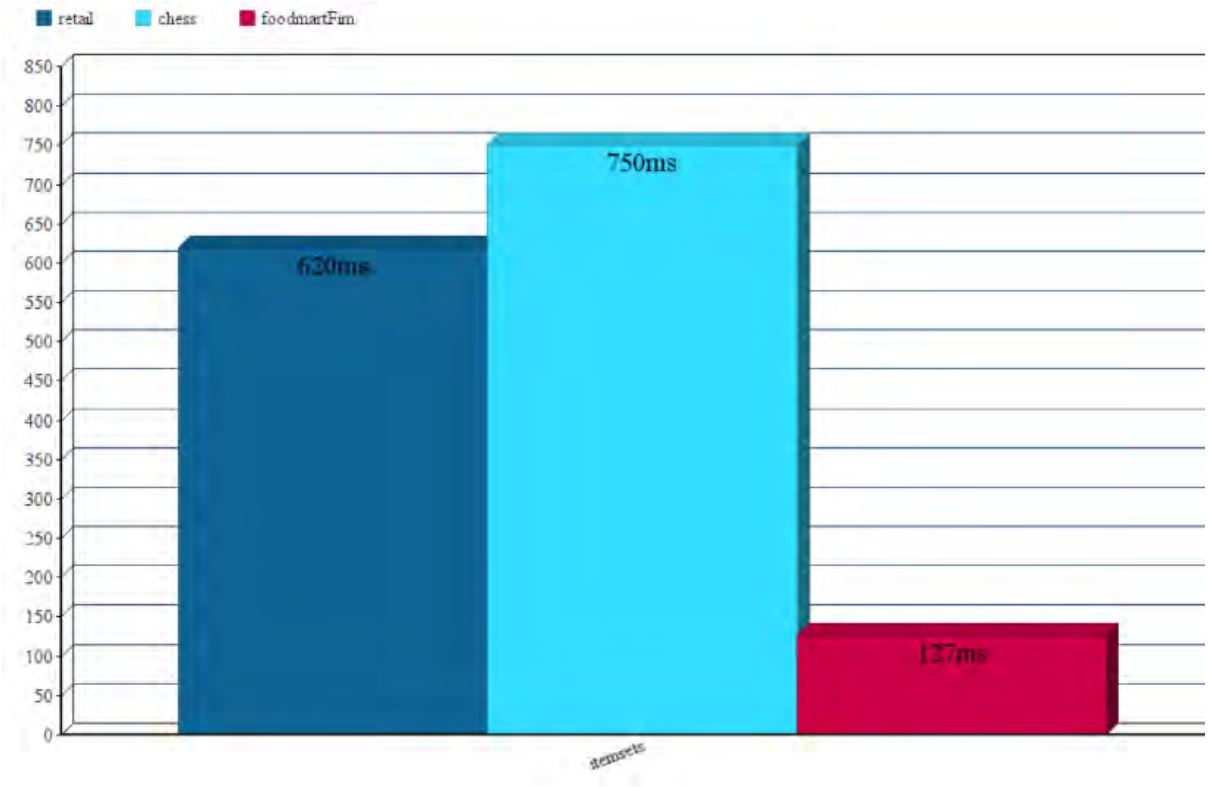

Figure 4.4.8: Runtime of FP Growth in different datasets

```
Plastic wrap Soda pop Wipes Melon #SUP: 2239
Plastic wrap Soda pop Formula Wipes Melon #SUP: 2239
Nuts Plastic wrap Wipes Melon #SUP: 2239
Nuts Plastic wrap Formula Wipes Melon #SUP: 2239
Nuts Plastic wrap Soda pop Wipes Melon #SUP: 2239
Nuts Plastic wrap Soda pop Formula Wipes Melon #SUP: 2239
Baby Wipes Melon #SUP: 2246
Plastic wrap Baby Wipes Melon #SUP: 2239
Plastic wrap Formula Baby Wipes Melon #SUP: 2239
Plastic wrap Soda pop Baby Wipes Melon #SUP: 2239
Plastic wrap Soda pop Formula Baby Wipes Melon #SUP: 2239
Nuts Plastic wrap Baby Wipes Melon #SUP: 2239
Nuts Plastic wrap Formula Baby Wipes Melon #SUP: 2239
Nuts Plastic wrap Soda pop Baby Wipes Melon #SUP: 2239
Nuts Plastic wrap Soda pop Formula Baby Wipes Melon #SUP: 2239
Formula Baby Wipes Melon #SUP: 2246
Soda pop Baby Wipes Melon #SUP: 2246
Soda pop Formula Baby Wipes Melon #SUP: 2246
Nuts Baby Wipes Melon #SUP: 2246
```

Figure 4.4.9: Frequent item set generation from FP Growth with minsup of 0.7

4.5 Analysis of Eclat algorithm:

Experiments are conducted for the analysis of the performances of ECLAT algorithm. The performance of the experiment is measured by comparing total execution time and total memory usage to execute the dataset in a proper manner. Eclat takes a depth first search approach (DFS). It uses a vertical database layout [8].

In vertical layout each item is represented by a set of transaction IDs (tidset), whose transaction contain the item. The primary idea of Eclat algorithm is to use the tid set intersection to compute the support of each candidate item set and to avoid the generation of subset that does not exist in prefix tree.[2]

Using tid list has an advantage that there is no need for counting support. The support of the item set is the size of the tidset representing it[10]. To perform the Eclat algorithm database is scanned few times. Virtual memory is needed to perform the operation. Eclat takes less execution time and memory than other two algorithms.

However, the main operation of Eclat is intersecting tidsets, thus the size of the tidset is one of the main factor that affect the running time and memory. While the tidset is large it takes more space to store the candidate set and it needs more time for intersecting tid sets.

4.6 Result of Eclat Algorithm:

Finally, we have performed Eclat algorithm for our initial dataset to mine the frequent pattern and predict the sales of a superstore. The initial dataset was large containing over three thousand transactions of seventy-five items containing in each transaction.

As it is said earlier that Eclat gives good run time and takes least memory storage if the tid list is small or medium. However, in this case the dataset was large and it takes more time to mining through the primary dataset we have used. But Eclat counted more frequent item set than the other two.

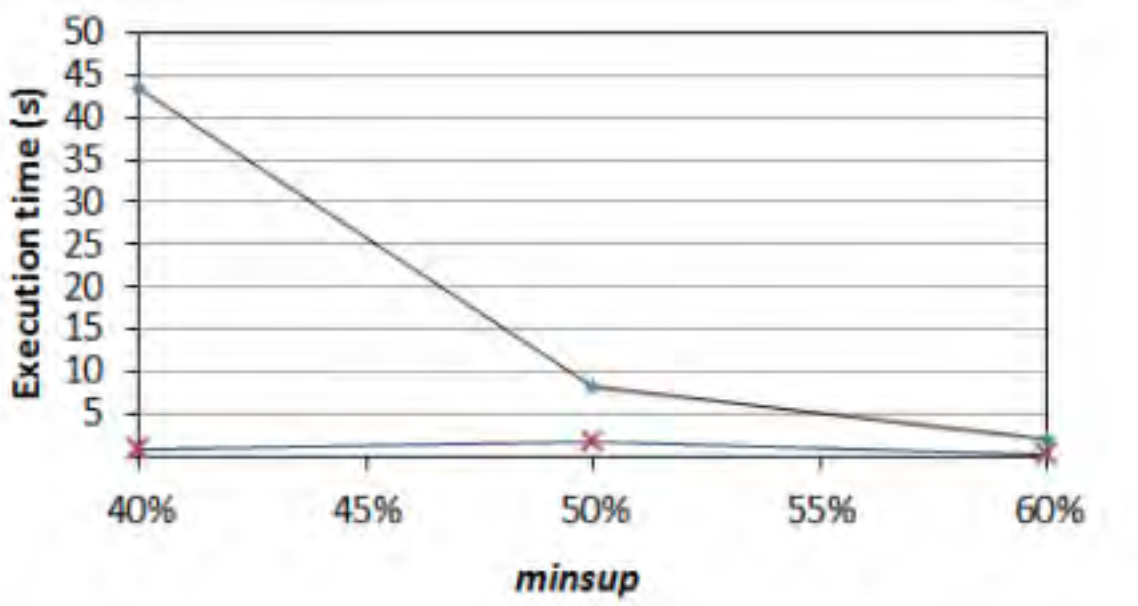

Figure 4.6.1: Performance graph of Eclat

```
=============  ECLAT v0.96r18 - STATS =============
 Transactions count from database : 3196
 Frequent itemsets count : 6439702
 Total time ~ 306549 ms
 Maximum memory usage : 1244.3535842895508 mb
------------------------------------------------------
```

Figure 4.6.2: Statistics of performance in chess.dat of Eclat algorithm

The figure shows the statistics and result of eclat performed upon chess.dat file. Previously, we got around forty eight thousands of frequent pattern from three thousand one hundred and ninety-six transactions. But here in eclat it gives us over six million frequent item sets of different size. As it is a large dataset and containing over six million item sets eclat took the most running time and memory in performing the execution. It took almost thirty minutes to produce the out put and consumes almost one gigabyte of memory.

To get more efficient result we have performed Eclat on several datasets. Previously when we preform FP Growth and Apriori on different dataset those two algorithms gave same frequent pattern result in every dataset though they vary from memory and run time issues. But Eclat gives different result specially in retail dataset. Eclat counts two frequent item sets in this phase. Whereas, other two did not count any.

```
------- FREQUENT ITEMSETS -------
 L0
 L1
 pattern 0:  40 support :  50675
 pattern 1:  49 support :  42135
 --------------------------------
============  ECLAT Retail Dataset - STATS ===
Transactions count from database : 88162
Frequent itemsets count : 2
Total time ~ 3625 ms
Maximum memory usage : 596.0500259399414 mb
================================================
```

Figure 4.6.3: Statistics of Eclat in retail dataset

Eclat nearly took six hundred MB of space and three minutes to generate this result and counts over eighty-eight thousand of transaction from this database.

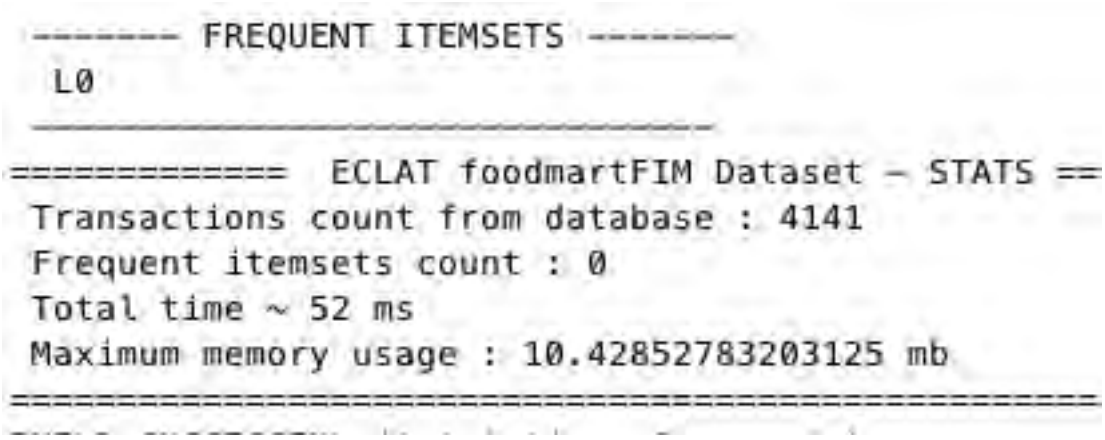

Figure 4.6.4: Statistics of eclat in foodmartFIM dataset

While testing the performance of eclat in another dataset foodmartFIM, eclat did not extract any set of item which is frequent. But takes 10 MB of memory which is a little bit greater than fp growth. In this case fp growth takes 127 milliseconds while eclat took only 52 milleseconds.

```
Candy Nuts Napkins Soda pop Whiskey Baby Berries Peaches #SUP: 1298
Candy Nuts Napkins Soda pop Whiskey Formula Baby Berries Peaches #SUP: 1297
Candy Nuts Napkins Whiskey Formula Baby Berries Peaches #SUP: 1308
Candy Napkins Whiskey Formula Baby Berries Peaches #SUP: 1316
Mushrooms Candy Whiskey Berries Peaches #SUP: 1331
Mushrooms Candy Plastic wrap Whiskey Berries Peaches #SUP: 1307
Mushrooms Candy Plastic wrap Soda pop Whiskey Berries Peaches #SUP: 1296
Mushrooms Candy Plastic wrap Soda pop Whiskey Formula Berries Peaches #SUP: 1295
Mushrooms Candy Nuts Plastic wrap Whiskey Berries Peaches #SUP: 1303
Mushrooms Candy Nuts Plastic wrap Soda pop Whiskey Berries Peaches #SUP: 1292
Mushrooms Candy Nuts Plastic wrap Soda pop Whiskey Formula Berries Peaches #SUP: 1291
Mushrooms Candy Nuts Plastic wrap Whiskey Formula Berries Peaches #SUP: 1302
Mushrooms Candy Plastic wrap Whiskey Formula Berries Peaches #SUP: 1306
Mushrooms Candy Soda pop Whiskey Berries Peaches #SUP: 1320
Mushrooms Candy Soda pop Whiskey Formula Berries Peaches #SUP: 1319
Mushrooms Candy Nuts Whiskey Berries Peaches #SUP: 1327
Mushrooms Candy Nuts Soda pop Whiskey Berries Peaches #SUP: 1316
```

Figure 4.6.5: Frequent itemsets in chess.dat datasets in eclat

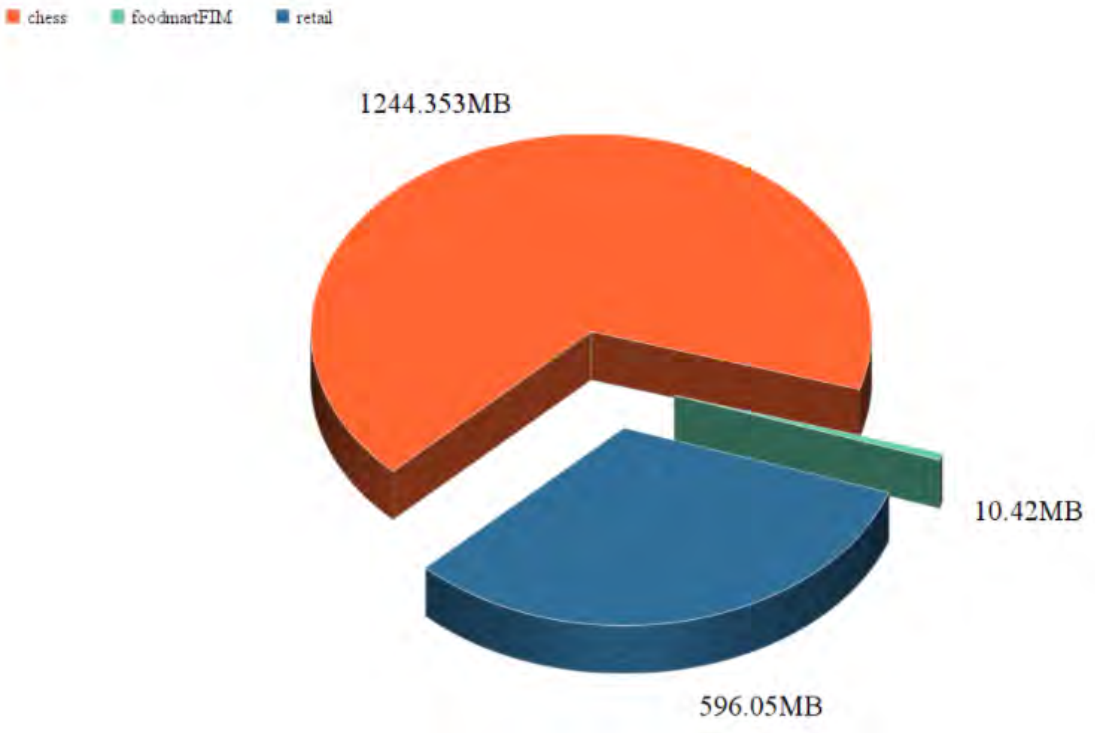

Figure 4.6.6: Memory Usage of eclat algorithm in different dataset

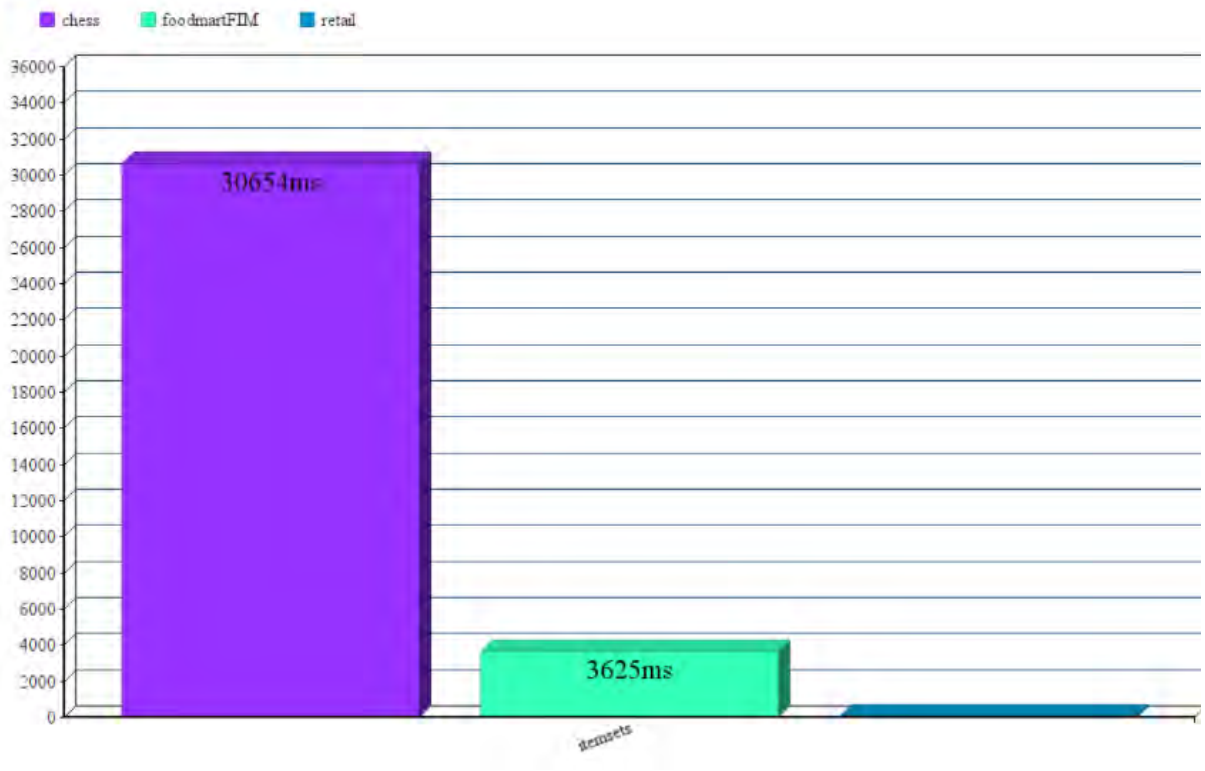

Figure 4.6.7: Runtime of eclat algorithm in different dataset

4.7 Comparative Analysis:

We have implemented all three algorithm– Apriori, FP Growth and Eclat in different dataset that contains transactional data of different supermarkets and predicted the most frequent item sets that the customers are buying regular basis. From this result the owner of the supermarkets can be benefitted. If they organize their product shelf according to the frequent pattern item set than sells will improve.

All these three algorithm have given quite a significant good result but there are some differences too. For example: apriori is the vey first algorithm for mining frequent patterns. It uses Breadth First Search Technique (BFS). It takes more execution time than other two

algorithm because it scannes the whole database each time a candidate item set is generated. Apriori uses horizontal layout of a databases.FP Growth is a tree based frequent pattern generating algorithm which uses divide and conquer method. It is applied in projected type database [2][4][5]. It does not need any candidate item set. Fp growth uses horizontal layout of the databases. It takes less time in comparison to apriori.

Eclat uses depth first search (DFS) for mining frequent patterns. The data is represented in bit matrix form in primary stage. It is faster than apriori and fp growth. It uses virtual memory to store data. Eclat does not scan database so frequently.

In our thesis, eclat performs well in terms of finding frequent patterns of item sets than fp growth and apriori. But eclat always takes much time when the transactional entries are larger. But fore medium and small dataset eclat is faster than fp growth and apriori.

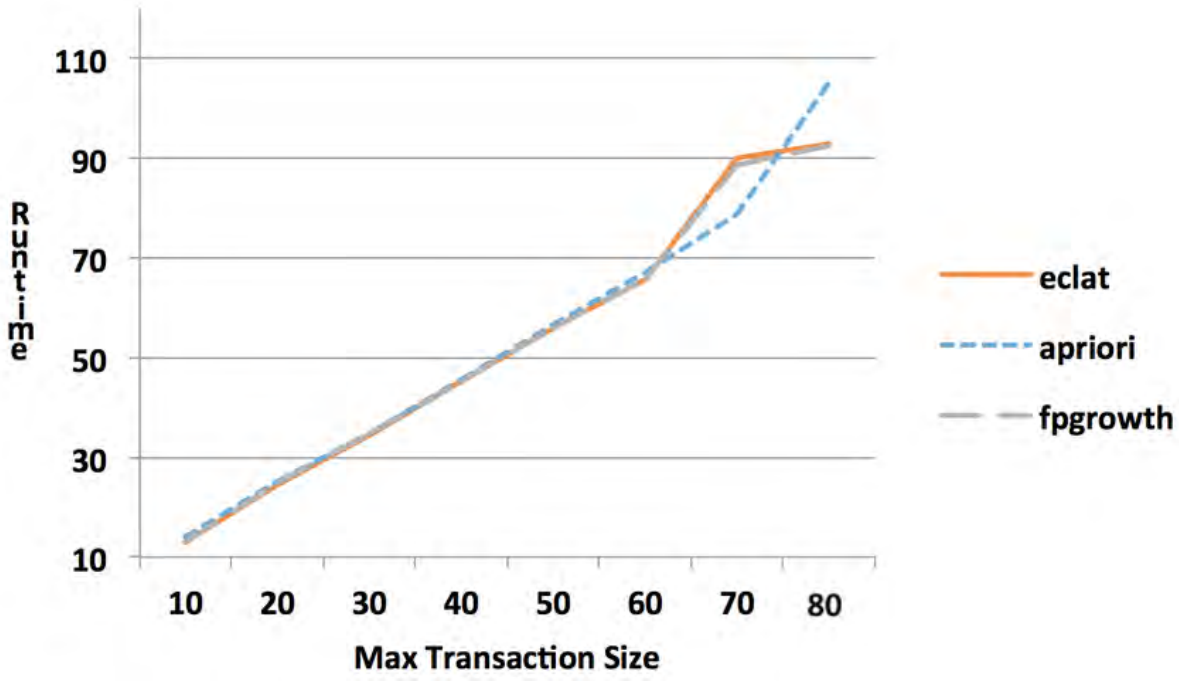

Figure 4.7.1: Runtime vs Max Transaction size comparison[12]

In case of memory usage eclat again took much memory while executing chess dataset. It takes 1.2 GB. While fp growth took 3.929 MB. In other dataset such as retail.dat 596 MB and 3625 mille seconds. While fp growth took 57.67 MB and 620 mille seconds.

Table 4.7.1 Memory Usage

Dataset	FpGrowth	Eclat
Chess.dat	**1.2 GB**	**3.929 MB**
Retail.dat	**596 MB**	**57.67MB**
foodmartFIM.dat	**10.42 MB**	**6.50 MB**

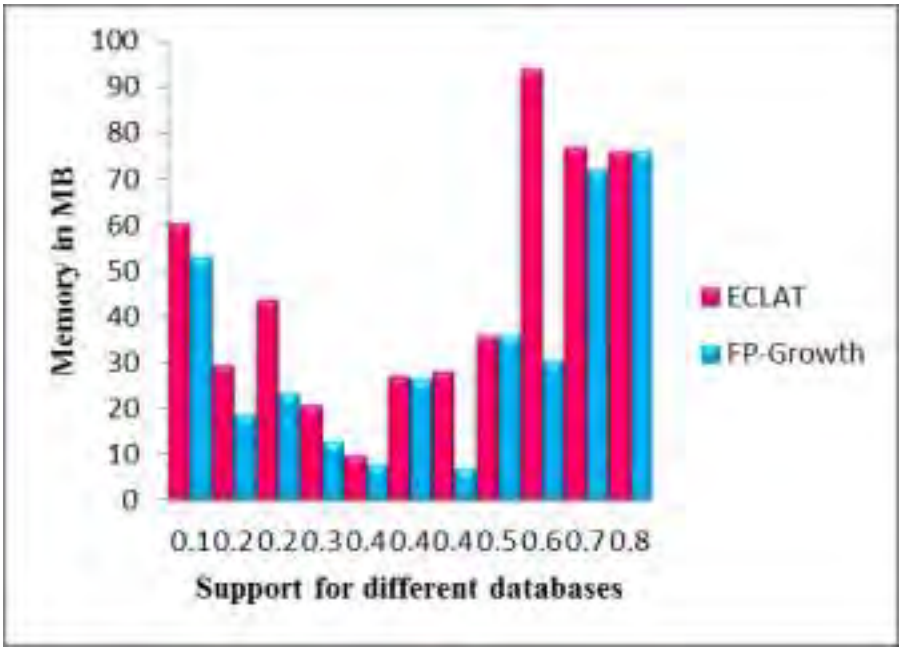

Figure 4.7.2: Memory Usage comparison between eclat and fp growth for different database [11]

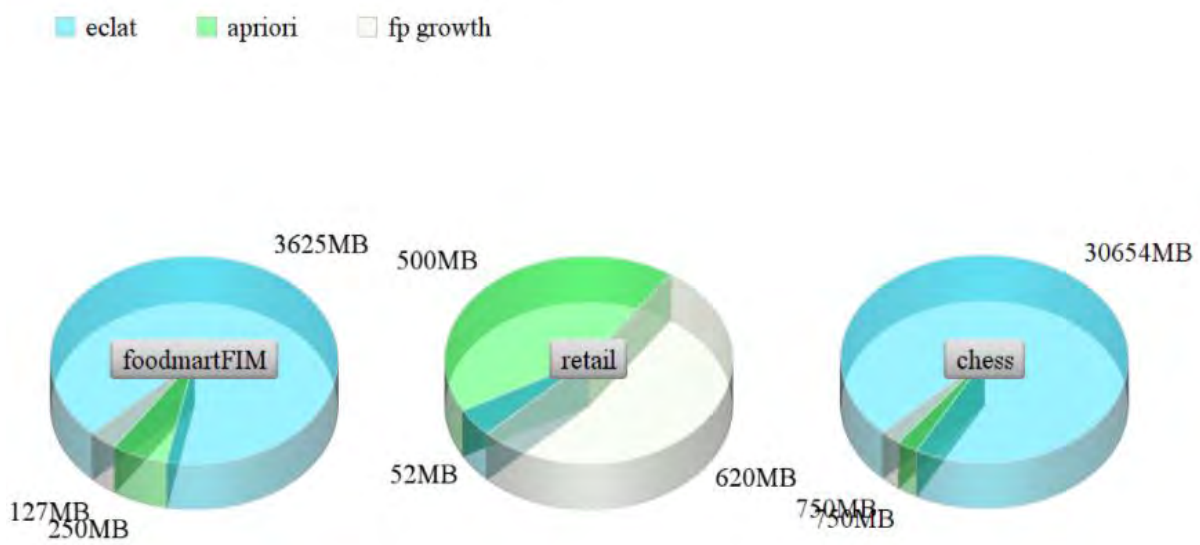

Figure 4.7.3: Memory Usage comparison among Eclat, FP Growth, Apriori in different dataset

Eclat is better in extracting frequent patterns among Apriori and FP growth. It generates two item set from retail data set from over eighty-eight thousand transactional count whilst Apriori and FP Growth did not find any frequent item set in retail database. In overall performance if we consider run time and storage as limit FP Growth is better. For large dataset with less support count Eclat outperform FP Growth and Apriori.

Following table[8] shows the theoretical differences among Apriori, FP Growth and Eclat algorithm.

Table 4.7.2: Table of comparison

Evaluation Criteria	Apriori	FP Growth	Eclat
1. Techniques	Breadth first search	Divide and Conquer	Depth first search and intersection of transaction id.
2. Database Scan	Database is scanned each time a candidate item set is generated	Database Id scanned two times only.	Database is scanned few times.
3. Advantages	-Easy to implement. -Use large item set property.	Database scanned two times only.	No need to scan database each time.
4. Disadvantages	-Require large memory space. -Too many candidate item set	FP tree is expensive to build consumes more memory.	It requires virtual memory to perform the transaction.
5. Data format	Horizontal	Horizontal	Vertical
6. Storage Format	Array	Tree (FP tree)	Array
7. Time	More execution time	Less time as compared to Apriori algorithm	Execution time is less than Apriori algorithm.

4.8 Recommendation:

After doing the result analysis we can come to the conclusion that for enormous dataset FP Growth performs better than Apriori and Eclat on the basis of time and memory usage. On the other hand, for small and medium dataset Eclat outperforms FP growth and Apriori on the basis of run time and memory space. So, we can recommend that a large organization which deals and maintain huge transaction database may use FP growth for accurate and fast result. Furthermore, organization which handles small or medium transaction database may use Eclat for improving their business strategy.

5.Conclusion:

Taking everything into account, by summing up the entire, we think this framework will have an efficient effect on marketing and sales analysis that can be used to make strategic business decision. This application can be extended to other fields like – sales tracking, product tracking, discount and pricing calculation etc. In future this method can be applied to very large databases where memory space is valuable and requires optimization. It can be further tuned for better performance and efficiency.

5.1: Limitations

The limitation first of all and most importantly we face about getting real life dataset. We have gone to every single big super shops of Bangladesh for their transaction data to conduct our thesis but they refused to give us. So with the help of our supervisor sir we work with artificial dataset.

5.2: Future work

In future we will try to implement new and advanced mining algorithm along with apriori, fp growth and eclat for better performance and fast result for sparse dataset. We will also build a user friendly web application where the user will be able to select as many items they wish and will be able to see the corresponding support count on a bar chart. From this they will be able to visualize and realize the relationship among particular items. The organizer of a super shop may use this application to arrange their products in the shop to boost their sell. Beside market basket data association analysis can be applied in other fields such as bio informatics, medical diagnosis and scientific data analysis

6.References:

[1] Saurabh Malgaonkar, Sakshi Surve and Tejas Hirave. “Use of Mining Techniques To Improve The Effectiveness of Marketing and Sales” IEEE Trans. Paper id-63.

[2] Mahmoud Houshmand, Mohammad Alishahi. “Improve the classification and Sales management of products using multi-relational data mining”. IEEE Journals,2011. 978-1-61284-486-2/111

[3] Xiaohui Yu, Yang Liu, Jimmy Xiangji Huang, Aijun An. “Mining Online Reviews for Predicting Sales Performance: A Case Studyin the Movie Domain”. IEEE Transactions On Knowledge and Data Engineering, VOL. 24, NO. 4, APRIL 2012.

[4] Julie Marcoux, and Sid-Ahmed Selouani. Universit«e de Moncton, Campus de Shippagan, Canada. “A Hybrid Subspace-Connectionist Data Mining Approach for Sales Forecasting in the Video Game Industry . 2009 World Congress on Computer Science and Information Engineering.

[5] Han, J., Kamber, M.: “Data Mining Concepts andTechniques”, Morgan Kaufmann Publishers, 2006.

[6] M.J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li,"New algorithms for fast discovery of association rules," in Third International Conference on Knowledge Discovery and Data Mining, 1997.

[7] M.J. Zaki, “Scalable Algorithm for Association Mining,” in IEEE TRANSACTION ON KNOWLEDGE AND DATA ENGINEERING, VOL. 12, NO. 3, MAY/JUNE 2000

[8] Rana I, Rathod A. Parul Institute of Engineering & Technology, Limba, Gujarat, India. “Frequent Item Set Mining In Data Mining: A Survey”, in Internaional journal on Recent and Innovation trends in Computing and Communication, Vol. 4, Issue.3

[9] Deepak Garg et. al. “Comparative Analysis of Various Approaches Used in Frequent Pattern Mining” (IJACSA) International Journal of Advanced Computer Science and Applications, Special Issue on Artificial Intelligence.

[10] Dr. Kanwal Garg, Deepak Kumar. Kurukshetra University,Haryana,India. “Comparing the performance of frequent pattern mining algorithm”, International Journal of Computer Applications (0975-8887), Vol 69- No. 25, May 2013.

[11] M. Sinthuja, N. Puviarsan, P. Aruna. Department of Computer Science And Engineering, Department of Computer Science And Information Sciences. Annamalai University. Chidanbaram, India. “Comparison Od Candidate Itemset Generation and Non Candidate Itemset Generation Algorithms in Mining Frequent Patterns”, International Journal on Recent and Innovation trends in Computing and Communication, Vol no 5, Issue:3.

[12] Heaton J.T. College of Engineering and Computing. Nova Southeastern University. Ft. Lauderable. “Comparing Dataset Characteristics that Favor the Apriori, Eclat or FP Growth Frequent Itemset Mining Algorithm”, conference paper,IEEE, Jan 2016.

[13] http://www.philippe-fournier-viger.com/spmf/index.php?link=algorithms.php

[14] Han J., Pei H., and Yin. Y., Mining Frequent Patterns without Candidate Generation, In Proc. Conf. on the Management of Data (2000)

[15] R. Agrawal, T. Imielinski, and A. Swami, ªMining Association Rules Between Sets of Items in Large Databases, ACM SIGMOD Conf. Management of Data, May 1993.

[16] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Inkeri Verkamo, ªFast Discovery of Association Rules, Advances in Knowledge Discovery and Data Mining, U. Fayyad and et al., eds., pp. 307-328, Menlo Park, Calif.: AAAI Press, 1996.

[17] R. Agrawal and J. Shafer, ªParallel Mining of Association Rules,IEEE Trans. Knowledge and Data Eng., vol. 8, no. 6, pp. 962-969, Dec. 1996.

[18] R. Agrawal and R. Srikant, ªFast Algorithms for Mining Association Rules, Proc. 20th Very Large Data Base Conf., Sept.1994.

[19] Rakesh Agrawal, Sakti Ghosh, Tomasz Imielinski, Bala Iyer, and Arun Swami. An interval classifier for database mining applications. In Proc. of the VLDB Conference, pages 560{573, Vancouver, British Columbia, Canada, August 1992.

[20] Wan Faezah Abbas, Nor Diana Ahmad, Nurlina Binti Zaini, "Discovering Purchasing Pattern of Sport Items Using Market Basket Analysis", International Conference on Advanced Computer Science Applications and Technologies, 2013.

[21] Pang-Ning Tan, Michael Steinbach, Vipin Kumar, Ïntroduction to Data Mining"