Research Article

A Novel Approach For Frequent Itemset Mining Using Geometric Progression Number Labeling

J. Ilamchezhian[1], Dr. V. Cyril Raj[2]

1. Research Scholar, Dr MGR Educational and Research Institute, Maduravoyal, Chennai-95
2. Professor, Dr MGR Educational and Research Institute, Maduravoyal, Chennai-95

Article History: Received: 10 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 28 April 2021

Abstract: The task of finding Frequent Item Sets is a very important and time-consuming process in the field of data mining. In the process of Knowledge discovery from data (KDD), the analysis of Frequent Itemset Mining (FIM) produces very use full information like how the items or data are correlated with each other and how frequently it is being found in the database especially in the transaction data, its report plays a vital role for the decision-makers to procure items and to stock it. Since the stock is the asset of the business and the profit of the business depends on the liquidity of the asset, more importance was given to Frequent Itemset mining. In this present Big data era, the variety and volume of data are growing tremendously; it is a challenging situation to mine the valuable information in a faster manner. Even though many researchers proposed many different kinds of algorithms for Frequent Item Set Mining it suffers defeat against either the size of frequent itemset or the time taken to finish the task. From this point of view, this paper proposes a new approach to mine the frequent items by using the Geometric progression series number as a label for each item and making its sum of items in the subset as a single value to find its frequency.

Keywords: Knowledge discovery from data (KDD), Data Mining, Frequent Itemset Mining (FIM), Big data, Geometric Progression (GP), Apriori, Eclat

1 Introduction

In the Business Analysis process of an organization, defining the business needs and recommendations are the major assignment to the top-level management to make a profit to stakeholders. These recommendations are usually given from the previous occurrences and events of the business. These kinds of data are collected daily from different sources. Data analytics plays a vital role in analyzing these data and bringing pieces of evidence from them. To claim the frequent occurrence of similar pieces of evidence over a certain period from the data, Association Rule Mining and Frequent Itemset Mining(FIM) are the essential tasks of data mining [11].

Data mining is the process of finding hidden patterns and useful knowledge from the data. This data mining has four main "super-problems" corresponding to clustering, classification, outlier analysis, and frequent pattern mining. Compared to the other three problems, frequent pattern mining is the most time-consuming process and it is directly proportional to the size of the data. When the patterns to be mined are very long, the number of subsets of frequent patterns can be extremely large. Over the last two to three decades, so many algorithms for mining frequent patterns in different perspective have been proposed by the researchers and still, the interest in this problem persists [9, 10].

The main focus of this paper is to identify frequent itemsets especially long frequent Itemsets at the earlier stage and this paper is organized as follows. Problem statement in section1, Different approaches in finding FIM in section 2, Discussion on various FIM Algorithms in section 3, Methods and Results in section 4 and conclusion with future work in section 5.

1.1 Problem Definition

Ruling out the relationships with the items present in the transactions of the database or dataset is the process of mining Frequent Patterns, which is stated as follows,

Given a database 'δ' with transactions '$\alpha_1 \ldots \alpha_n$' from which Frequent Itemset mining will determine all patterns 'γ' which present in at least a fraction 'μ' of the transaction or transactions. This fraction 'μ' is referred as minimum support.

Symbols used in definition: δ – Database, α-Transaction, β-Item, λ-Frequent Item Set, μ-minSupport, γ- Pattern

Definitions:

Let, $\alpha = \{ \alpha_1, \alpha_2, \ldots, \alpha_n\}$ be a transactions of database δ, where each $\alpha_i \in \alpha$, $\forall i = \{1 \ldots n\}$ consists of a set of items, say $\alpha_i = \{\beta_1, \beta_2, \beta_3, \ldots \beta_m\}$ where β_i is an Item. A set $\gamma \subseteq \alpha_i$ is called an itemset or a pattern. The size of an itemset is defined by the number of items it contains. If the size of the itemset is k then that itemset can be represented as k-itemset. The number of transactions containing γ is referred to as the support of γ and when it exceeds minimum support μ it is considered to be frequent in the database. Table 1.1 is a sample database with 20 transactions.

Table 1.1: Items List

Item Code	Item Name
ITC_1	KNIFE,
ITC_2	HIGHLIGHTER,
ITC_3	PEN,
ITC_4	SKETCH
ITC_5	PENCIL,
ITC_6	WHITENER
ITC_7	ERASER
ITC_8	POUCH
ITC_9	PAPER

Table 1.2: Transaction database with Item names.

TRANSACTION NO.	TRANSACTION ITEMS
T_XN1	KNIFE, PENCIL, WHITENER, POUCH
T_XN2	HIGHLIGHTER, SKETCH, POUCH
T_XN3	SKETCH, PENCIL, ERASER
T_XN4	HIGHLIGHTER, PEN
T_XN5	PENCIL, WHITENER, ERASER
T_XN6	HIGHLIGHTER, PEN, SKETCH
T_XN7	HIGHLIGHTER, WHITENER, ERASER, PAPER
T_XN8	PENCIL
T_XN9	POUCH
T_XN10	PEN, PENCIL, ERASER
T_XN11	PEN, PENCIL, ERASER
T_XN12	PENCIL, WHITENER, POUCH
T_XN13	HIGHLIGHTER, PEN, ERASER
T_XN14	PENCIL, WHITENER, ERASER
T_XN15	HIGHLIGHTER, SKETCH
T_XN16	SKETCH, POUCH
T_XN17	SKETCH, PENCIL
T_XN18	KNIFE, WHITENER, POUCH
T_XN19	PEN, PENCIL, ERASER
T_XN20	HIGHLIGHTER, WHITENER, ERASER

Table 1.3: Transaction database with Item codes

.

TRANSACTION NO.	TRANSACTION ITEMS
T_XN1	ITC_1, ITC_5, ITC_6, ITC_8
T_XN2	ITC_2, ITC_4, ITC_8
T_XN3	ITC_4, ITC_5, ITC_7
T_XN4	ITC_2, ITC_3
T_XN5	ITC_5, ITC_6, ITC_7
T_XN6	ITC_2, ITC_3, ITC_4
T_XN7	ITC_2, ITC_6, ITC_7, ITC_9

T_XN8	ITC_5
T_XN9	ITC_8
T_XN10	ITC_3, ITC_5, ITC_7
T_XN11	ITC_3, ITC_5, ITC_7
T_XN12	ITC_5, ITC_6, ITC_8
T_XN13	ITC_2, ITC_3, ITC_7
T_XN14	ITC_5, ITC_6, ITC_7
T_XN15	ITC_2, ITC_4
T_XN16	ITC_4, ITC_8
T_XN17	ITC_4, ITC_5
T_XN18	ITC_1, ITC_6, ITC_8
T_XN19	ITC_3, ITC_5, ITC_7
T_XN20	ITC_2, ITC_6, ITC_7

Table 1.4: Candidate Itemsets with its support.

Itemsets	Frequency/support Count
{}	0
{ITC_1}	2
{ITC_2}	7
{ITC_3}	6
{ITC_4}	6
{ITC_5}	10
{ITC_6}	7
{ITC_7}	9
{ITC_8}	6
{ITC_9}	1
{ITC_2,ITC_3}	3
{ITC_2,ITC_4}	3
{ITC_2,ITC_5}	0
{ITC_2,ITC_6}	2
{ITC_2,ITC_7}	3
{ITC_2,ITC_8}	1
{ITC_3,ITC_4}	1

Itemsets	Frequency/support Count
{ITC_3,ITC_5}	3
{ITC_3,ITC_6}	0
{ITC_3,ITC_7}	3
{ITC_3,ITC_8}	0
{ITC_4,ITC_5}	1
{ITC_4,ITC_6}	0
{ITC_4,ITC_7}	1
{ITC_4,ITC_8}	2
{ITC_5,ITC_6}	4
{ITC_5,ITC_7}	6
{ITC_5,ITC_8}	2
{ITC_6,ITC_7}	4
{ITC_6,ITC_8}	3
{ITC_7,ITC_8}	0
{ITC_3, ITC_5, ITC_7}	3
{ITC_5, ITC_6, ITC_7}	0
{ITC_6, ITC_7, ITC_8}	0

Table 1.5: Frequent Itemsets with its support.

Itemsets	Frequency/support Count
{ITC_2}	7
{ITC_3}	6
{ITC_4}	6
{ITC_5}	10
{ITC_6}	7
{ITC_7}	9
{ITC_8}	6
{ITC_2,ITC_3}	3
{ITC_2,ITC_4}	3
{ITC_2,ITC_7}	3
{ITC_3,ITC_5}	3
{ITC_3,ITC_7}	3
{ITC_5,ITC_6}	4
{ITC_5,ITC_7}	6
{ITC_6,ITC_7}	4
{ITC_6,ITC_8}	3

[ITC_3, ITC_5, ITC_7]	3

2 Different approaches in Frequent Itemset Mining.

2.1 Frequent Itemset

A Frequent pattern $\lambda \subseteq \gamma$ is called a Frequent itemset [1] where ‘γ’ is a set of items or patterns. Any set of items is said to be a frequent itemset when it occurs at least in the minimum number of transactions ‘α’ of a database ‘δ’ and when which exceeds minimum support ‘μ’.

From Table1.2, considering 15% as minimum support ‘μ’ for the database ‘δ’, 3 will be the support count for the said database. The itemsets in Table1.5 are said to be Frequent Itemset since its count exceeds the minimum support ‘μ’.

This process of finding frequent itemsets have been proposed by many researchers in many ways. The icon approaches are discussed in section 2.3 and section 3.

2.2 Frequent Itemset mining

Finding such a Frequent Itemsets ‘λ’ from the database ‘δ’ is called Frequent Itemset Mining. The Computational Challenge in finding Frequent Itemsets from the transactional database made a high level of interest for the researchers. Even for a moderate-sized dataset, the search space of (FPM) Frequent Pattern Mining is enormous, which is exponential to the length of the transactions in the dataset. When the minimum support is low then itemset generation process challenges the researchers. In fact, in most practical scenarios, the support levels at which one can mine the corresponding itemsets are limited by memory and computational constraints.

2.3 Approaches in finding Frequent Itemsets.

To improve the efficacy of the frequent pattern mining most of the researchers used different approaches in which AIS is the first algorithm proposed by Agrawal etl., Imielinski etl. andSwami etl. and later it was improved as the Apriori algorithm[1] by R. Agrawal etl. and R. Srikant etl. Further, the pitch of finding frequent itemsets is followed in different directions and approaches with the major goal of reduced execution time, improved efficiency and economic usage of main memory.

FIM Algorithms are broadly classified based on the measures of interestingness [6].

1. Threshold / Support -based frequent itemset mining algorithms
2. Non-Support-based frequent itemset mining algorithms

However, most of the algorithms have been developed with the mindset of following

- Reducing the number of passes over the database [6,3]
- Mining without Candidate generation [3]
- Reducing the Candidate key size using Hashing technique [4]
- Pruning the candidates by different techniques [7,8]
- Partitioning database to reduce the size to fit in memory [5]
- Using multiple threshold or Minimum supports, Local support count and Global support Count [5]
- Using the Labeling approach with Prime numbers, GCD numbers and with some Weights [7,8]

3 Discussion on famous Algorithms

3.1 Apriori algorithm

Apriori[1] introduced the generate-and-eliminate approach, which consists of iteratively generating a set of candidate itemsets, then computing its frequency in the database. In the elimination phase, each iteration requires a pass on the complete database, when the frequency of the candidate itemset exceeds the user-specified minimum support count then they have resulted as Frequent Item Set and those sets are used to construct the candidate itemsets for the next pass. Since this algorithm uses downward closure property, the succeeding execution will be stopped when there are no more Candidates to generate.

Using this Apriori algorithm from the database ‘δ’ and its transactions as given in Table 1.3, candidate Itemsets generated in all passes with its frequency of occurrence in the database are given in Table 1.4. The Frequent Itemsets pruned in all the 3 passes are given in Table 1.5. Thus totally of 17 frequent itemsets are found for the given minimum support ‘μ’ of 15% from the given database ‘δ’.

The major disadvantages of this algorithm are (i) the number of passes over the entire database in eliminating phase which means, to find k-itemset this algorithm requires the same k-passes over the database. And some times, most of the generated candidates may not be frequent and occupying more main memory. From the Table1.4, it was found that {ITC_2,ITC_5}, {ITC_3,ITC_6}, {ITC_3,ITC_8}, {ITC_4,ITC_6}, {ITC_7,ITC_8}, {ITC_5, ITC_6, ITC_7}, {ITC_6, ITC_7, ITC_8} are the itemsets having its frequency count as 0. So it is unnecessary to generate such candidate sets. Because of these issues, FIM on big-sized sparse or temporal databases may take hours or even days to complete in finding Frequent Item Sets.

3.2 Partition Algorithm

The transaction database is read twice in this Partition algorithm, irrespective of the minimum support and irrespective of the number of partitions. This is a major notable advantage when compared to the Apriori algorithm. The Partition algorithm uses the same technique of candidates itemset generation of the Apriori algorithm. The cost of subset operation per itemset in the latter passes decreases due to the decrease in the length of the candidate's list in the Partition algorithm when compared with the Apriori algorithm.

Partition Algorithm[5] has two phases and each phase reads the database once. First, the database is divided into many partitions which are non-overlapping and each of its partition can be held in the main memory. Generates its all local large itemsets or otherwise called Super Sets from each partition and generates global candidate itemset by combining all local large itemsets from all partitions having the same length. In the next phase, it counts the support of each global large itemset in the database.

Even though the Partition algorithm performs better than Apriori, the sizes of the local and global candidate sets may be susceptible to data skew and a few larger itemset may present commonly in more than one partition leads to a false global candidate set which also depends on how the partition was carried out in the database.

3.3 Eclat Algorithm

The Eclat [2] algorithm uses the Lattice traversal technique and the structural property of itemset to identify the frequent itemsets especially large ones and their subsets. These subsets are organized such that they can be fit and solved in the main memory. Searching techniques like top-down or bottom-up or both with fewer database scans makes the algorithm more efficient than the former algorithms.

Table 3.1: Vertical layout of transactions with Item and its support.

Item Code	Item Name	Count
ITC_1	T_XN1,T_XN18	2
ITC_2	T_XN2, T_XN4, T_XN6, T_XN7, T_XN13, T_XN15, T_XN20	7
ITC_3	T_XN4, T_XN6, T_XN10, T_XN11, T_XN13, T_XN19	6
ITC_4	T_XN2, T_XN3, T_XN6, T_XN15, T_XN16, T_XN17	6
ITC_5	T_XN1, T_XN3, T_XN5, T_XN8, T_XN10, T_XN11, T_XN12, T_XN14, T_XN17, T_XN19	10
ITC_6	T_XN1, T_XN5, T_XN7, T_XN12, T_XN14, T_XN18, T_XN20	7
ITC_7	T_XN3, T_XN5, T_XN7, T_XN10, T_XN11, T_XN13, T_XN14, T_XN19, T_XN20	9
ITC_8	T_XN1, T_XN2, T_XN9, T_XN12, T_XN16, T_XN18	6
ITC_9	T_XN7	1

Eclat performs significantly better than Apriori and Partition algorithms and Eclat makes only a few databases scans, requires no hash trees, and uses only simple intersection operations to generate frequent itemsets. Further Eclat can handle lower support values in dense datasets, where both Apriori and Partition run out of virtual memory at 0.25% support. Even though candidate generation is eliminated in the Eclat algorithm, it suffers in the transformation of data from the Horizontal database to the Vertical database and the algorithm is particularly effective when the discovered frequent itemsets are long.

3.4 FP-Tree Algorithm

In the view of Candidate set generation, both Apriori and Partition algorithms the generation of the huge number of candidate sets is inevitable. For example, if there are 10^4 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^7 candidate 2-itemsets. Regarding database scanning both Eclat and Partition algorithms take two database scanning, Apriori takes more database scans to find Frequent Itemset.

To overcome these issues the frequent-pattern growth, FP-growth[3] uses a divide-and-conquer strategy and mines the complete set of frequent itemsets in a single database scan, without candidate generation and any data transformation like Horizontal database to Vertical database layout.. First, this FP-growth algorithm compresses the database into FP-Tree (Frequent Pattern Tree)with itemset relationa and its association information. The algorithm creates conditional database for each association of frequent itemset and each conditional database is mined separately. Frequent Item header table is constructed to facilitate traversal in FP-tree then each item points to its correlated items of the set present in each transaction through its node links as child nodes. By this way the whole databse is represented in the form of FP-tree. Firstly, 1K-frequent itemsets are scanned and made as an initial suffix pattern and then a conditional pattern base was constructed. A sub-database with a set of prefix paths that is co-occurring with the same suffix pattern in the FP-tree is created and then a conditional FP-tree is constructed. It is continued recursively until the entire tree was searched. By concatenating the suffix patterns with frequent patterns which are generated from the conditional FP-tree, pattern growth was achieved. The FP-growth method substantially reduces the searching cost of the long frequent patterns by selecting the least frequent items as a suffix pattern.

4 Methods and Results discussion

The chess dataset was downloaded from the philippe-fournier-viger.com website and was used. The database size is 335Kb with 3196 transactions and 75 items which includes 37 maximal transaction [12]. The Spmf software tool downloaded from the philippe-fournier-viger.com website [13] and it was used to check the performance of the algorithms Apriori, Eclat and FP Tree in a laptop with the configuration of processor Intel i5 7200 U, 2.5 GHz with 4GB ram. These algorithms are measured in terms of memory Occupancy, Execution time and Data Storage Space occupied in the hard disk. From Table 4.1 the Apriori algorithm consuming high memory when compared with other algorithms.

The Eclat algorithm is consuming nearly the same amount of memory and the FP tree algorithm consuming relatively high memory for the least minimum support when compared with the Eclat algorithm. From Table 4.2 it was clear that the Apriori algorithm is performing worst at least minimum support and FP tree performance was better at high minimum support. Eclat algorithm performance was overall better than the other two algorithms even at least minimum support. From Table 4.3, it is a good measure that the Apriori algorithm does not require more hard disk space while running whereas the FP tree algorithm occupying a very high amount of hard disk space during the execution of the algorithm.

Data storage space is gradually increasing as the minimum support is decreasing. On consideration of these findings, this paper proposes a new methodology with less execution time, less memory and less hard disk storage space.

minSup%	Apriori	Eclat	FP Tree
5	720	370	587
10	507	366	512
15	310	361	472
20	231	356	445
25	215	350	426
30	447	364	392

Table 5.1: Memory occupied during execution

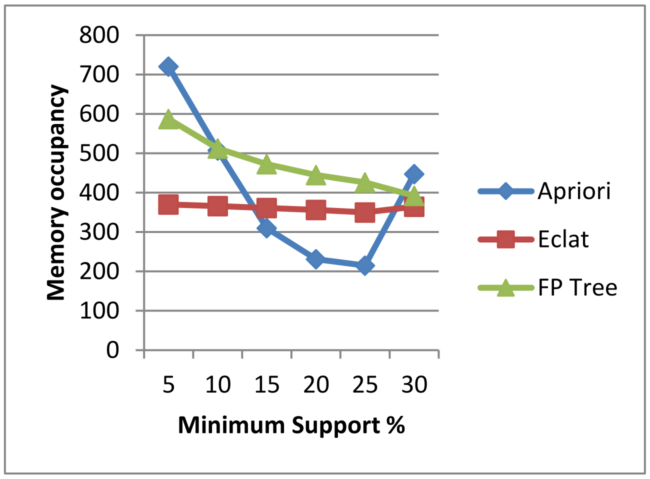
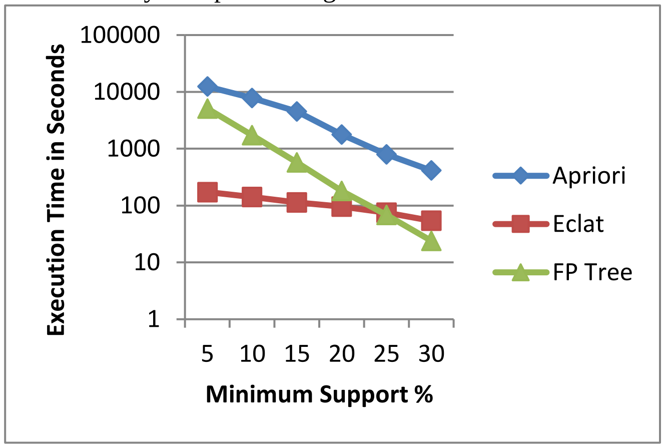


Figure.1: Memory occupied during execution

minSup%	Apriori	Eclat	FP Tree
5	12425	173	5096
10	7791	142	1732
15	4503	114	577
20	1786	96	181
25	792	75	68



30	**414**	**54**	**24**

Table 5.2: Algorithm Execution time in seconds **Figure.2**: Execution time in seconds

minSup%	**Apriori**	**Eclat**	**FP Tree**
5	**56**	**6553.6**	**308224**
10	**39**	**5427.2**	**110592**
15	**25.6**	**4403.2**	**39220**
20	**17.5**	**3584**	**10752**
25	**12.4**	**2867.2**	**3482**
30	**8.7**	**2252.8**	**1332**

Table 5.3: Data storage in HDD

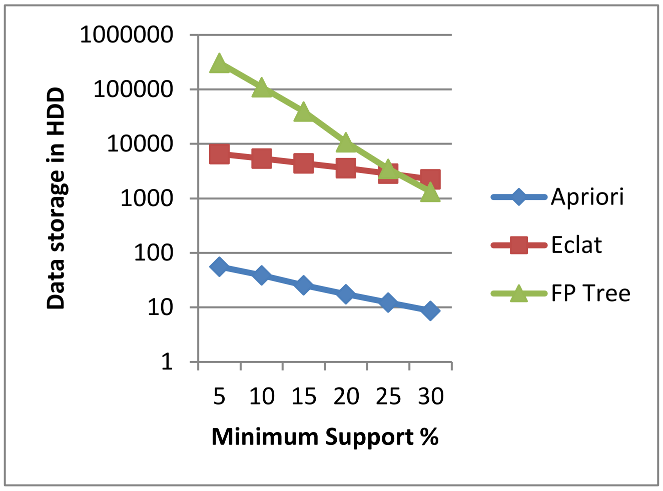


Figure.3: Data storage in HDD

5 Proposed method

From the results of Apriori, Eclat and FP tree algorithms, it was clearly showing that the Apriori algorithm takes more execution time when compared to the other two algorithms, but Eclat consumes more memory than the former one. FP tree algorithm is better than the other two algorithms but that data storage is very high. Both Eclat and FP tree algorithm fail to perform for big data. Because the entire data has to be kept in the main memory, practically it is not possible, so it leads to out of memory space error. Thus these three algorithms are not fit for big data with normal computing facility. Though there are hundreds of algorithms were proposed by many researchers are the hybrids of the above tree algorithms or combination of those algorithms and all have their limitations.

This paper proposes a new approach to find frequent itemsets using the straight forward method. Geometric Progression Number(GPN) is generated and that is used for labelling. The GPN is generated using the following formula.

$$GPLN_n=a*r^{n-1}$$ … **Formula-1**

Where, a - First Element of the series, r – Any constant number, n – the nth element of the series. In this proposed method, 1 is taken for 'a', 2 is taken for 'r' and 'n' is the number of items starting from 1.

The GPN thus created are unique numbers and the sum of GPN will not come as a result as another GPN in the series. Because of the nature of this GPN, so created in such a way was used in this methodology. Firstly the singleton frequent itemsets are found similar to the Apriori algorithm in the first pass of the database and the singleton infrequent itemsets are eliminated. Now each frequent singleton itemset is assigned with a unique GPN as a label. Then transactions are read one by one in the second pass of the database the infrequent itemsets are discarded. The rest of the items in the transaction are now mapped with GPN which was already assigned to the items as labels. As the next step, those items present in their transaction after the elimination of infrequent items as a superset, all its subsets are found. For each subset created the corresponding GPN are summed up as a single value and labelled with that calculated GPN for finding its frequency of occurrence in the transaction database. For the transactions of the database δ in Table 1.1, the GPN is assigned as in Table 5.1.

Table 5.1: GPN is assigned as a label for the singleton Frequent Itemsets.

Item Code	GPN
ITC_1	Infrequent
ITC_2	1
ITC_3	2
ITC_4	4
ITC_5	8
ITC_6	16
ITC_7	32
ITC_8	64
ITC_9	Infrequent

A transaction α1 is read from the database δ and its items {β1, β2, β3, . . . βm} is replaced with GPNs. The first transaction from Table1.3 is replaced with GPN as follows.

$$\begin{tikzcd} \text{ITC\_1,} \arrow[d] & \text{ITC\_5,} \arrow[d] & \text{ITC\_6,} \arrow[d] & \text{ITC\_8} \arrow[d] \\ \text{Infrequent} & 8 & 16 & 64 \end{tikzcd}$$

All the subsets of the transactions are found after the removal of the infrequent itemsets. Since all the singleton frequent itemsets are already found, starting from Itemset size of 2 succeeding subsets are generated as follows.

{ ITC_5, ITC_6 } , { ITC_5, ITC_8 } , { ITC_6, ITC_8 } , { ITC_5, ITC_6, ITC_8 }

Then its corresponding GPNs are mapped for items and its sum was found. So a single value will be found for an itemset or subset. For the above subsets, the sum of GPNs will be as listed in Table 5.2

Table 5.2: Calculated GPN for the subsets of the transaction database.

Subset	Calculate GPN	Subset	Calculate GPN
{ ITC_5 , ITC_6 }	8 + 16 = 24	{ ITC_2 , ITC_6 }	1 + 16 = 17
{ ITC_5 , ITC_8 }	8 + 64 = 72	{ ITC_2 , ITC_7 }	1 + 32 = 33
{ ITC_6 , ITC_8 }	16 + 64 = 80	{ ITC_6 , ITC_7 }	16 + 32 = 48
{ ITC_5, ITC_6 , ITC_8}	8 + 16 + 64 = 88	{ ITC_2 , ITC_6 , ITC_7 }	1 + 16 + 32 = 49
{ ITC_2 , ITC_4 }	1 + 4 = 5	{ ITC_3 , ITC_5 }	2 + 8 = 10
{ ITC_2 , ITC_8 }	1 + 64 = 65	{ ITC_3 , ITC_7 }	2 + 32 = 34
{ ITC_4 , ITC_8 }	4 + 64 = 68	{ ITC_2 , ITC_3 }	1 + 2 = 3
{ ITC_2 , ITC_4 , ITC_8 }	1 + 4 + 64 = 69	{ ITC_3 , ITC_5 , ITC_7 }	2 + 8 + 32 = 42
{ ITC_4 , ITC_5 }	4 + 8 = 12	{ ITC_2 , ITC_3 , ITC_7 }	1 + 2 + 32 = 35
{ ITC_4 , ITC_7 }	4 + 32 = 36	{ ITC_5 , ITC_6 , ITC_7 }	8 + 16 + 32 = 56
{ ITC_5 , ITC_7 }	8 + 32 = 40	{ ITC_2 , ITC_3 , ITC_4 }	1 + 2 + 4 = 7
{ ITC_4 , ITC_5 , ITC_7 }	4 + 8 + 32 = 44	{ ITC_3 , ITC_4 }	2 + 4 = 6

Now each calculated value is taken as a label and its frequency was initialized with 1. So the candidate set as follows

{ 72 , 1 } , { 24 , 1 } , { 80 , 1 } , { 88 , 1 }

In this candidate set, the first portion is the calculated value and the next is its count or frequency. Similarly, all the subsets of all the transactions are found and the candidates are generated. If the generated candidate was already found then the count will be increased by 1 otherwise count is initialized by 1. So the final found candidates are listed in Table 5.3

Table 5.3: candidate subsets generated from the transactions using calculated GPNs

Calculated GPN label	Frequency	Calculated GPN label	Frequency
3	3	40	6
5	3	42	3
6	1	44	1
7	1	48	4
10	3	49	2
12	2	56	2
17	2	65	1
24	4	68	2
33	3	69	1
34	4	72	2
35	1	80	3
36	1	88	2

Table 5.4: Frequent Itemsets with its frequency of occurrence and calculated GPN

Calculated GPN label	Frequency	Orginal Subset
3	3	{ ITC_2 , ITC_3 }
5	3	{ ITC_2 , ITC_4 }
10	3	{ ITC_3 , ITC_5 }

24	4	{ ITC_5 , ITC_6 }
33	3	{ ITC_2 , ITC_7 }
34	4	{ ITC_3 , ITC_7 }
40	6	{ ITC_5 , ITC_7 }
42	3	{ ITC_3 , ITC_5 , ITC_7 }
48	4	{ ITC_6 , ITC_7 }
80	3	{ ITC_6 , ITC_8 }

Table 5.5: Likely to be frequent Itemsets

Calculated GPN label	Frequency	Orginal Subset
12	2	{ ITC_4 , ITC_5 }
17	2	{ ITC_2 , ITC_6 }
49	2	{ ITC_2 , ITC_6 , ITC_7 }
56	2	{ ITC_5 , ITC_6 , ITC_7 }
68	2	{ ITC_4 , ITC_8 }
72	2	{ ITC_5 , ITC_8 }
88	2	{ ITC_5, ITC_6 , ITC_8}

To reduce the memory occupancy, the unique found label with its representation of itemsets are store in the file, which will be used only in the final stage. When compared to the cost of memory, the cost of the I/O operation will be less.

In this proposed method, unnecessary candidates like in the Apriori algorithm or not generated. Because the candidates are generated from the items present in the transaction. Considering the minimum support specified by the user is 15 % then the minimum support count will be 3. So from Table 5.3, those items which exceed the threshold or minimum support count of 3 is picked out as frequent itemsets and listed in Table 5.4. For the found frequent itemsets are represented in the form of calculated GP n and so that has to be replaced by the original itemsets as shown in Table 5.4. Thus found frequent itemsets are combined with the singleton frequent itemset which was found already and produced as final frequent itemsets.

The major advantage of this proposed method is:

- Not like the Apriori algorithm, this method eliminates unwanted items itemsets
- It takes only a single pass over the database to find all the frequent itemset after finding singleton frequent itemsets like the Apriori algorithm and FP tree algorithm.
- Frequent itemsets of any size can be found straightforwardly in a single scan of the database.
- Since all the itemsets are stored in the file and using only their simple label as a candidate the amount of memory occupancy will be less when compared to the Eclat or FP tree algorithms.
- The amount of data stored is also lesser than the FP tree algorithm data storage space.
- Execution time is lesser than the apriori algorithm execution time due to the fewer number of passes over the database.

Another important advantage in this proposed method is Likely To Be Frequent Itemset in Table 5.5 also can be found from the same data. Because those Likely To Be Frequent Itemset also be useful information for the decision-maker to improve the business in the Market-Basket based problems. This method is implemented and tested with the Test database given in Table.1.2 and compared the performance with the famous algorithms Apriori, Eclat and FP tree.

The following are the findings:

- The execution time is lesser than the Apriori algorithm.
- Memory consumption is more or less equal to the Eclat algorithm
- Data storage space is more or less equal to the FP tree algorithm.

The limitations found in this proposed method are there GPN are found in a 2^n pattern. So approximately 62 items can be assigned with the GPN and when finding the calculated GPN for bigger itemsets it may exceed the 64 bit of memory space. Considering 2^{63} is the range of value for an integer. These limitations are to be further evaluated in the future work of the research.

6 Conclusion

From the famous algorithms and approaches in finding frequent item sets discussed in this paper, we conclude with the advantages and disadvantages of these algorithms. Though the Apriori is the straight forward algorithm that takes multiple scanning of the database for the size(k) of the frequent itemset, so the execution time is very high. Even though the Eclat algorithm is faster than the Apriori algorithm as it requires few passes over the database and insensitive to data-skew, the number of items reflecting in few transactions may reduce its performance. And when the database having more items and huge numbers of transactions, it is not possible to keep the entire (vertical) database in the main memory. But the use of a partitioning algorithm may solve this issue and some researches were also addressed this fact. Though FP-Tree is the fast algorithm and which does not generate candidate, the amount of consumed memory is usually much more as compared to Eclat but when the dataset or database size is large it is time-consuming because it has to construct the FP-Tree for the whole database and it may fails when there is not enough memory to keep the entire database in the main memory. As the algorithm follows top-down and depth-first search techniques in Tree traversal for finding frequent patterns consumes more time with the database having long transactions with less support count.

The use of labels and representation of transactions and their subsets in short form has reduced the memory usage than the Eclat and FP Tree algorithms when tested with the test database. As the proposed method in this paper has less computational complexity, further implementing this idea in big data and real-time databases will speed up the process of identifying the frequent itemset.

Reference

1. R.Agrawal, R.Srikant, “Fast algorithms for mining association rules”, 20th VLDB Conference Santiago Proceedings, Chile, 1994
2. MJ.Zaki, “Scalable Algorithms for Association Mining”. IEEE Transactions on Knowledge and Data Engineering, 12(3), May/Jun 2000, pp.372-390.
3. Jiawei Han, Jian Pei, Yiwen Yin, Runying Mao, “Mining frequent patterns without candidate generation A frequent-pattern tree approach”, Data Mining and Knowledge Discovery, 8, 53–87, 2004,Kluwer Academic Publishers.
4. Jayasuruthi L,Shalini A,Vinoth Kumar V.,(2018) ” Application of rough set theory in data mining market analysis using rough sets data explorer” Journal of Computational and Theoretical Nanoscience, 15(6-7), pp. 2126-2130
5. Ashok Savasere, Edward Omiecinski, Shamkant Navathe, “An Efficient Algorithm for Mining Association Rules in Large Databases”, Proceedings of the 21st VLDB Conference Zurich, Switzerland, 1995
6. P.N. Tan, V. Kumar, and J. Srivastava. “Selecting the Right Interestingness Measure for Association Patterns.” ACM KDD Conference, 2002.
7. S.N.Sivanandam,Dr.S.Sumathi, T.Hamsapriya, K.Babu, “Parallel Buddy Prima – A Hybrid Parallel Frequent itemset mining algorithm for very large databases”, Academic Open Internet Journal Volume13, 2004
8. Gawwad, M A, Ahmed, M F and FayekM B, “Frequent Itemset Mining for Big Data Using Greatest Common Divisor Technique” Data Science Journal, 16: 25, 2017 ,pp. 1–10, DOI: https://doi.org/10.5334/dsj-2017-025
9. B. Goethals. “Survey on frequent pattern mining” Technical report, University of Helsinki, 2003.
10. Bart Goethals, ”Data Mining and knowledge discovery”- handbook
11. Jiawei Han Micheline Kamber, Jian Pei, “Data Mining Concepts and Techniques”, 3rd Edition Morgan Kaufmann , 2011
12. http://www.philippe-fournier-viger.com/spmf/datasets/chess.txt
13. http://www.philippe-fournier-viger.com/spmf