Communications on Applied Nonlinear Analysis
ISSN: 1074-133X
Vol 32 No. 1 (2025)

Design and Analysis of JIC Algorithm on Big Data

Ilamchezhian J[1], Kannan A[2], Cyrilraj V[3], N. Padmapriya[4]

[1]Associate Professor, Dr. M.G.R. Educational and Research Institute, Maduravoyal, Chennai-95, India.

[2] Former Professor, College of Engineering, Anna University, Guindy, Chennai – 25, India.

[3]Professor, Dr. M.G.R. Educational and Research Institute, Maduravoyal, Chennai-95, India.

[4]Assistant Professor, Department of Statistics, Sri Sarada College for Women (Autonomous), Salem, TN, India

[1]chezhian.ilam@gmail.com

[2]akannan123@gmail.com

[3]cyrilraj@drmgrdu.ac.in

[4]theen.91@gmail.com

Article History:

Received: 12-11-2024

Revised: 15-12-2024

Accepted: 11-01-2025

Abstract: Computers have significantly impacted various fields, but managing vast amounts of information remains challenging. Artificial intelligence helps machines make data-driven decisions, but large datasets still pose difficulties for researchers. To tackle challenges and gain deep insights from large datasets, we proposed a novel technique using Geometric Progression series numbers (GPLN) for labeling singleton frequent itemsets and Cumulative Geometric Progression series numbers (CGPLN) for labeling itemsets with multiple frequent items. Initially, the algorithm used 2 as the constant 'r' for generating the series, but that was inadequate for large datasets. So, this paper proposes the Jagged Itemset Counting (JIC) algorithms 1 and 2 by reducing the value of 'r' and introduced dotted pairs for CGPLN labels to represent frequent itemsets. This redefined methodology requiring two passes over the transaction database: the first pass involves pre-processing, identifying singleton (1-k) frequent itemsets, determining GPLN, and partitioning the dataset. The partitions are processed sequentially: JIC-Algorithm-1 is applied to the first partition and JIC-Algorithm-2 to the remaining partitions. The n-k frequent itemsets from all partitions are combined using the Join algorithm, alongside 1-k itemsets found earlier. For small and medium-sized data, the JIC methodology outperforms Apriori and Eclat algorithms, showing better execution time even at low support thresholds. In Big Data scenarios, while FP-Growth and Eclat struggled, the proposed methodology excelled in execution time, main memory consumption, and disk memory utilization.

Keywords— Frequent Itemset Mining (FIM), Big Data, Jagged Itemset Counting (JIC), Geometric Progression (GP), Geometric Progression Label Number (GPLN), Cumulative Geometric Progression Label Number (CGPLN), Apriori, Eclat, FP Growth.

Introduction

The introduction of computing technology has greatly impacted business data processing and scientific computing. Initially, data storage was limited, but storage capacity has exponentially grown, doubling every two years. Today, organizations accumulate massive volumes of data, stored in megabytes, gigabytes, or terabytes daily [3]. The HACE theorem outlines the key characteristics of big data, which is essential for various business applications and analyses. Frequent itemset mining is a crucial method for identifying common data patterns. In our earlier

works, the initial method for mining frequent items involved using Geometric Progression series numbers as labels for each item, summing the items in subsets to find their frequency [1]. This approach used the constant 'r' as 2 for generating the series and the generated labels are inadequate for large datasets. To overcome this limitation, the Jagged Itemset Counting (JIC) algorithm was introduced. This algorithm employs Geometric Progression sequence numbers for labeling items, making it easier to mine frequent items. The Geometric Progression Label Number (GPLN) labels individual items, while the Cumulative Geometric Progression Label Number (CGPLN) labels itemsets [2]. This straightforward method enhances the speed of Frequent Itemset mining, leading to the development of the JIC methodologies for mining frequent itemsets from voluminous data. Despite numerous algorithms developed over the past few decades, many struggle with identifying frequent itemsets from sparse datasets at low support counts and efficiently handling large datasets. Among the seven V's of big data, 'Volume' is the most challenging for frequent itemset mining. As the JIC algorithm was a straightforward method that speeds up the Frequent Itemset mining process, it was further modified and named the Jagged Itemset Counting methodologies to mine frequent itemsets from voluminous data.

This paper addresses this by introducing Jagged Itemset Counting methodologies, pre-processing raw internet data, and reading the database twice. It uses six algorithms: Pre-Processing, Partitioning, GPLN, Jagged Itemset Counting (JIC) - 1, JIC - 2, and Join operation, with performance measured in execution time and primary memory consumption.

I. BACKGROUND AND LITERATURE

FIM algorithms are classified into two categories: those based on threshold/support measure and those without. The first algorithm to find Association-rules was proposed in 1993 and later renamed as the Apriori algorithm [4] [5]. The Apriori algorithm, Eclat algorithm [6], and FP growth algorithm [7] are the roots of most FIM algorithms. However, the Apriori algorithm may take longer to execute when the data set size is large. The Eclat algorithm uses a depth-first search technique but may not be effective for sparse databases. The FP-Tree algorithm loads data into main memory and creates a prefix-based tree representation. The FP Growth algorithm offers benefits but has drawbacks, such as complexity, cost, and memory requirements for large datasets.

The Partition [8] reads transaction databases twice without considering partitions, but has disadvantages like partition size skew and poor performance when common super sets are present in multiple partitions. DHP [9] is a hash-based technique used to efficiently minimize candidate itemset size in early stages, with faster execution time than Apriori but longer hash table generation time. The Apriori method, initially refined and renamed Dynamic Itemset Counting (DIC), performed poorly in some datasets due to its low support level and infrequent items [10]. For mining frequent itemsets from dynamic datasets using ZIGZAG backtracking search and dynamically maintained support information without database scanning was introduced [11]. The effectiveness of this strategy is influenced by the distribution of items in transactions, as it does not yield impressive results when data is skewed [12].

The Buddy Prima algorithm [13], which uses prime numbers to represent transactions, but requires more computational power, memory, and I/O processing capacity for large databases like VLDB. A new methodology was introduced using weights for frequent itemsets, based on user-specified minimum range [14]. The runtime also depends on the weight range. PRISM approach was developed for frequent sequence mining, using primal-block encoding notation for itemsets, but its cardinality is insufficient for massive data sets [15]. A Bi-Eclat technique was introduced for improving the real Eclat algorithm, but found it unstable in sparse databases and inadequate for massive ones [16]. FP-Tree-based [17] infrequent itemset mining was

proposed by adjusting pattern size based on computational complexity, with run time increasing with higher lower-bound values. AprioriMin [18] was introduced using a support value approximation method for frequent itemset discovery, but found it underperforms when support is minimal. The N-list [19] approach under the Subsume technique have significantly improved the pre-post methodology for locating frequent itemsets, but not for sparse data.
HashEclat [20] uses an approximation strategy for optimizing the Eclat algorithm using MinHash, but its effectiveness depends on matrix-sample-rate, "wrong-estimation" and "wrong omission". WFRIM [21] mines FIs using a user-specified weight and threshold, requiring more processing time and memory when the regularity threshold has more variations. The Single Scan (SS) approach [22] used for finding frequent itemsets using High Performance Computing, but this method faces memory constraints. A new method for mining frequent-itemsets by trimming transactions iteratively from lattice, which showed performance issues with smaller dataset sizes and limited distinct item counts [23].
For identifying frequent item sets from big data in distributed computing settings by highlighting the significant increase in execution time with database size [24]. The authors present an innovative approach to Frequent Itemset Mining, which overcomes the challenges of existing methods and provides fresh insights from various data sources.

II. PRELIMINARIES AND PROBLEM STATEMENT

Let the supermarket transaction database, TxDb, contains the set of transactions. $\mathrm{Txn} = \{\mathrm{Txn}_1, \mathrm{Txn}_2, \dots, \mathrm{Txn}_n\}$ where $\mathrm{Txn}_i \in \mathrm{Txn}, \forall i = \{1, 2, \dots, n\}$. TxDb is a database made up of a countable number of Items, which are represented as $\mathrm{TxI} = \{\mathrm{TxI}_1, \mathrm{TxI}_2, \dots, \mathrm{TxI}_n\}$, where $\mathrm{TxI}_i \in \mathrm{TxI}, \forall i = \{1, 2, \dots, n\}$. A transaction, Txn_i, is a combination of items bought by a single customer. As a result of $\mathrm{Txn}_i \subseteq \mathrm{TxI}$. An Itemset γ containing one or more items such that $\gamma \subseteq \mathrm{Txn}_i$ and whose size is determined by the number of items contained within it. A set is said to be a frequent itemset when its support count φ exceeds the minimum support count σ_{min}, calculated using the user-specified threshold σ. The support count φ is calculated by counting the number of occurrences of items in the TxDb transaction.
Definition-1 (GPLN): A Geometric Progression Label Number (GPLN) is a mathematical Geometric Progression sequence number produced by Equation 1.

$$\mathrm{GPLN} = a * r^{n-1} \quad \dots \mathrm{Eqn.}\,(1)$$

where 'a' is the first term, 'r' is a constant and 'n' is the sequence's n^{th} term. The final term 'n' represents the total number of unique items in the TxDb. The JIC algorithm calculates 'a' as ten times the number of unique items (numItems*10) in the database TxDb and 'r' is calculated using Equation 2.

$$r = 1 + \frac{9.25}{a} \quad \dots \mathrm{Eqn.}\,(2)$$

where a=numItems*10. 1 and 9.25 are the constants.

Definition-2 (CGPLN): In each transaction, the Cumulative GPLN for each subset γ will be calculated by simply adding all the GPLN of the subset's corresponding items. The CGPLN is calculated using Equation 3 below.

$$\mathrm{CGPLN}(\gamma) = \sum_{\mathrm{TxI}_i \in \gamma} \mathrm{GPLN}(\mathrm{TxI}_i)$$
$$\text{where } \mathrm{TxI}_i \in \gamma, \forall i = \{1, 2, \dots, |\gamma|\} \quad \dots \mathrm{Eqn.}\,(3)$$

Definition-3 (CGPLN-Label): It is the label representation for the itemsets and it is assigned after the CGPLN of each itemset is calculated using the label structure shown in Equation 4. This label structure, in the style of three-dotted values, is used to label the itemsets uniquely (xxxx.xxxx.xxxx). These three dotted values represent the CGPLN-calculated value, the number of items in the Itemset and the sum of the item-values respectively, as shown in the syntax below:

$$\text{CGPLN} \,.\, \text{Itemset Size} \,.\, \text{Sum of Items} \qquad \ldots \text{Eqn. (4)}$$

Definition-4 (Frequent Itemset): The singleton (1-k) frequent itemsets discovered as found in the Apriori algorithm are identified during the first phase of the algorithm execution (Agrawal and Srikant 1994). The n-k frequent itemsets are discovered based on its support count. When the support φ of CGPLN-Label exceeds the minimum support count $\sigma_{\min}$, the itemsets are said to be frequent n-k itemsets.

$$\varphi(\text{CGPLN} - \text{Label}) > \sigma_{\min} \qquad \Longrightarrow \qquad \text{CGPLN} - \text{Label is Frequent}$$

The CGPLN-Labels are replaced with their corresponding original itemsets and added to the final frequent itemset list during the Join algorithm implementation.

III. JAGGED ITEMSET COUNTING (JIC) ALGORITHMS AND METHODOLOGIES

The real-time dataset is downloaded from the website# and that cannot be used directly for data mining. To understand the buying pattern from sales as well as customer purchasing behaviour from the transaction database, the following are the algorithms which are designed as shown in the Figure-1 for this proposed work: algorithm for Pre-Processing, algorithm for partitioning the dataset, algorithm for GPLN, algorithm for Jagged Itemset Counting (JIC) -1, algorithm for Jagged Itemset Counting (JIC) -2 and algorithm for Join.

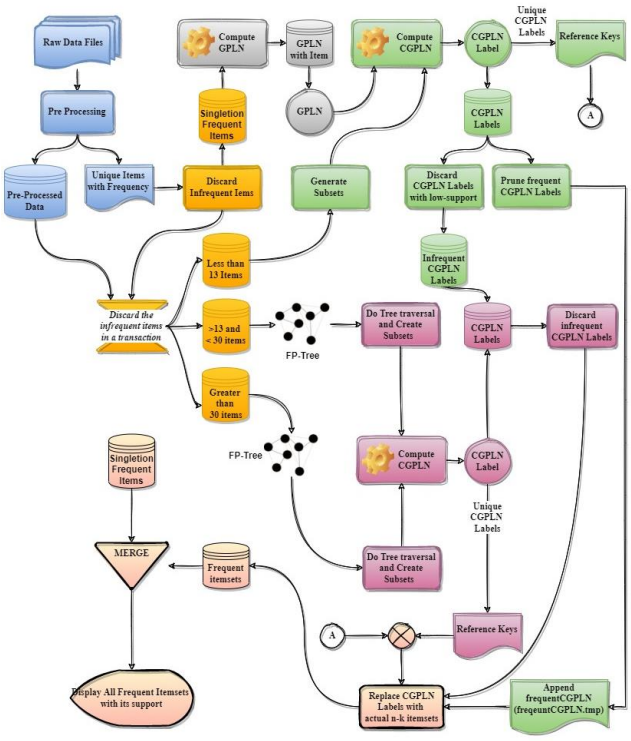

Fig. 1 System Architecture diagram

The raw data downloaded from the website is pre-processed in such a way that all the items associated with a given order-id are aggregated into a single row, with the reference id. Order-ids that are redundant are removed. Columns and fields that are irrelevant to this study are also removed. Table 1 displays the order details for the given raw dataset, with order id in the first column and product id in the second. All the items associated with a specific order id are grouped into a single row separated by a comma with that order id referenced. This data is then inserted into the pre-processed file as a single row.

Algorithm for Pre-Processing:

1. $Initialize\ all\ the\ variabes$
2. $Open\ Raw-data\ (TrData.csv)\ file$
3. $Read\ a\ Line\ from\ TrData\ and\ split\ into\ lineParts\ with\ Delimiter\ 'Comma'$
4. $TrDataCounter++$
5. $ordID_i \leftarrow lineParts[orderID]$
6. $pSequence \leftarrow Append\ lineParts[productID]$
7. $Read\ NextLine\ from\ TrData\ and\ split\ into\ lineParts\ with\ Delimiter\ 'Comma'$
8. $TrDataCounter++$
9. $ordID_j \leftarrow lineParts[orderID]$
10. $If\ ordID_j == ordID_i then$
11. $\quad pSequence \leftarrow Append\ lineParts[productID]$
12. $\quad If\ not\ TrData.EOF\ then$
13. $\quad\quad Continue\ from\ Step-7$
14. $\quad End\ if$
15. $Endif$
16. $Write\ pSequence\ in\ preDb.dat\ file$
17. $preDbCounter++$
18. $pSequence = null$
19. $If\ not\ TrData.EOF\ then$
20. $\quad Continue\ from\ Step-3$
21. $Else$
22. $\quad Exit$
23. $End\ if$
24. $Close\ TrData\ file$
25. $Write\ Metadata\ file\ with\ numTransactions\ and\ numItems$

A. *Partitioning*

According to the literature, the number of candidate set subsets will grow at an exponential rate. In view of this and in order to improve performance in frequent itemset mining, the pre-processed file is further partitioned into three Partitions based on the number of items present in each transaction. The unique items are already identified during the pre-processing with its support and now the items with its support less than the 'GlobalSupportCount' will be removed and that are considered as infrequent item.

$$GlobalSupportCount\ \ \sigma_{min}^{G} = Number\ of\ Trnsactions * (\sigma\ /100) \quad \ldots \text{Eqn. (5)}$$

From the each transaction of the pre-processed file, infrequent items are removed and the singleton frequent items are counted. The partition is chosen based on the number of items in a transaction line and data is written into the corresponding dataset. Partition-1 will only include transactions containing 13 or fewer items, Partition-2 will include transactions

containing more than 13 but less than or equal to 30 items and Partition-3 will include transactions containing more than 30 items. The Pseudo code for the partitioning algorithm as follows:

Algorithm for Partitioning

1. $Initialize\ all\ the\ variables$
2. $Open\ the\ database\ preDb.dat$
3. $Read\ the\ database\ preDb$
4. $For\ each\ transaction\ Txn_i\ in\ preDb$
5. $itemCount++$
6. $numTransactions++$
7. $Split\ Txn_i\ into\ parts\ with\ delimiter\ 'Space'$
8. $if\ partsCount > 30\ then$
9. $Write\ Txn_i in\ preDb3.dat$
10. $numTransactions3++$
11. $Elseif\ partsCount > 13\ then$
12. $Write\ Txn_i in\ preDb2.dat$
13. $numTransactions2++$
14. $Else$
15. $Write\ Txn_i in\ preDb1.dat$
16. $numTransactions1++$
17. $Endif$
18. $Find\ support\ for\ each\ item\ TxI_i$
19. $itemsCollection \leftarrow unique\ TxI_i$
20. $Endfor$
21. $Close\ preDb.dat, preDb1.dat, preDb2.dat\ and\ preDb3.dat\ files$
22. $GlobalSupportCount\ \ \sigma_{min}^{G} \leftarrow threshold * numTransactions$
23. $For\ each\ Item\ TxI_i\ in\ itemsCollection$
24. $If\ \varphi(TxI_i) \geq \sigma_{min}^{G}\ then$
25. $FIS_{1k} \leftarrow add\ TxI_i$, *FIS –Frequent Item Set*
26. $End\ if$
27. $End\ for$
28. $Write\ FIS\ in\ singletonFIS.dat\ with\ item'sFrequency$
29. $Write\ metaData.dat\ with\ numTransactions,$

$numTransactions1$, *numTransaction2,numTransaction3,numItems and Global Support Count.*

TABLE I

ORDER DETAILS OF RAW DATA MADE BY THE CUSTOMERS

order_id	product_id	add_to_cart_order	reordered
1	49302	1	1
1	11109	2	1
1	10246	3	0
1	49683	4	0
1	43633	5	1
1	13176	6	0

1	47209	7	0
1	22035	8	1
36	39612	1	0
36	19660	2	1
36	49235	3	0

B. Geometric Progression Label Numbering

The singleton frequent items are read from the previously saved list of frequent items, in order to calculate the Geometric Progression Label Number (GPLN). Each singleton frequent item was given a Geometric Progression Label Number using the procedure outlined in Equation 1. These GPLNs are stored for later usage, together with the representative item names that correspond with them. The pseudo code for the GPLN algorithm is as follows:

Algorithm for Partitioning

1. $Initialize\ all\ the\ variables$
2. $Open\ singletonFIS.dat$
3. $a = numitems * 10$
4. $r = (1 + 9.25)/a$
5. $flag = true, x = 0$
6. $For\ each\ Item\ TxI_i\ in\ singletonFIS$
7. $\quad If\ flag\ then$
8. $\quad\quad n = 1 + x$
9. $\quad\quad flag = false$
10. $\quad Else$
11. $\quad\quad n = numItems - x$
12. $\quad\quad flag = true$
13. $\quad End\ if$
14. $\quad GPLN = a * r^{n-1}$
15. $\quad x++$
16. $\quad\quad GPLN - List \leftarrow GPLN(TxI_i)$
17. $End\ for$
18. $Write\ GPLN\ with\ ItemName$

C. Jagged Itemset Counting (JIC) Stage -1

In this procedure, a transaction line was read from the PreDb1 dataset using the first split of the preprocessed dataset. Every subset of a transaction is generated, with the exception of the empty set and the subset with a single item. During the partitioning phase, we have already eliminated the infrequent elements from the transaction to minimize computing complexity. Each subset item was then mapped from their GPLN list to its corresponding GPLN, which was determined and stored using the GPLN method. We used Definition-2 to calculate the cumulative GPLN. As described in Definition-3, a CGPLN-Label representation is made for every subset and stored in a text file for future use.

$$\varphi(CGPLN - Label) > GlobalSupportCount\ \sigma_{min}^{G} \quad \text{... Eqn. (6)}$$

$$\varphi(CGPLN - Label) > \frac{1}{6}\sigma_{min}^{G} \quad \text{... Eqn. (7)}$$

For Every new CGPLN-Label, its count was assigned by 1, and the count of any existing label was increased by 1. When compared to other algorithms like Apriori, this technique will reduce needless procedures like searching and sorting the items or itemsets contained in the transaction.

It will also lower the execution time. Until every transaction is read, this process is repeated. When CGPLN-Labels support more than the 'GlobalSupportCount' during this procedure, they are categorized as frequent CGPLN-Labels and are stored in a hash map. While CGPLN-Labels with balance are rare, those with support larger than one-sixth of 'GlobalSupportCount' are deemed 'Likely to be frequent CGPLN-Labels' and are stored in a temporary file for potential future frequency. The residual CGPLN-Labels that are rare are eliminated.

Algorithm for Jagged Itemset Counting (JIC)-1

1. $Initialize\ all\ the\ variables$
2. $Open\ the\ database\ preDb1.dat$
3. $Read\ the\ database\ preDb1$
4. $For\ each\ transaction\ Txn_i\ in\ preDb1$
5. $Remove\ infrequent\ items$
6. $Find\ all\ subsets\ of\ Txn_i$
7. $For\ each\ subset\ \gamma_j\ in\ Txn_i$
8. $CGPLN - Label_j \leftarrow CGPLN(\gamma_j).|\gamma_j|.sum(\gamma_j)$
9. $If\ \varphi(CGPLN - Label_j) = 0\ then$
10. $CGPLN - Labels \leftarrow CGPLN - Label_j$
11. $Write\ \gamma_j@CGPLN - Label_j\ in\ ReferenceKeys.tmp\ file$
12. $CGPLN - Label_j - count = 1$
13. $Else$
14. $CGPLN - Label_j - count++$
15. $End\ if$
16. $End\ for$
17. $End\ for$
18. $For\ each\ CGPLN - Label_j\ in\ CGPN - Labels$
19. $If\ \varphi(CGPLN - Label_j) < \sigma_{min}^{G} then$
20. $if\ \varphi(CGPLN - Label_j) \geq {}^{1}/_{6}\ (\sigma_{min}^{G})\ then$
21. $Write\ \ CGPLN - Label_j with\ its\ support\ into\ inFrequent.tmp$
22. $Discard\ CGPLN - Label_j$
23. $Else$
24. $Write\ \ CGPLN - Label_j with\ its\ support\ into\ frequent.tmp$
25. $End\ if$
26. $End\ for$
27. $Close\ preDb1.dat$

D. Jagged Itemset Counting (JIC) Stage -2

This process is invoked when the partitions have transactions with more than 13 items. Initially, the infrequent CGPLN-Labels which that were stored during the previous Jagged Itemset Counting (JIC) Stage-1 process are loaded, and their counts are updated. The transactions from these partitions are read one by one. After eliminating the infrequent item from each transaction, the frequent items are re-ordered based on their support count in such a manner that the item with the least support will go last and items with higher support will come first. The transactions are stored in a transaction collection, and the first item with the highest support from each transaction is stored in the Header-List representing that it is a starting node.

Form transaction collection: a node is created for each item in the transaction if it does not exist in the existing node list and its count is set to 1. The count of the node was incremented by 1

and updated if it already existed in the node list. The node's parent and child connections are built from the header node to the leaf node for each transaction. A pattern tree was created using the header node with its parent-child relationship from the whole transaction collection, so that the path of each node in the Header-List is then tracked using parent and child links. Then, for every header node of the pattern tree and its related parent-child path, a set was created, and its subsequent subsets are stored in the list.

Using Definitions 1, 2 and 3, the GPLN, CGPLN, and CGPLN-Labels are found for every subset, and its support count was found using the previous methodology and is updated in a hash map. Now the frequent CGPLN-Labels are moved, and their support counts are updated in the final frequent list, which was created in the previous JIC-Stage-1 process. The infrequent CGPLN-Labels and its support count are now merged with the previously stored 'Likely to be frequent CGPLN-Labels'. After merging, the frequent CGPLN-Labels are moved to the final frequent list, and the pattern tree will be purged. From the rest of the CGPLN-Labels, when the support of CGPLN-Label exceeds the local support count (σ_{min}^{Local}), they are moved to the 'Likely to be frequent CGPLN-Labels' collection after purging the existing old collection, and they are written back to the infrequent CGPLN-Label temporary file. This process is continued when there are more partitions to proceed, and the final frequent list, the 'Likely to be frequent CGPLN-Labels' collections and their support counts are updated subsequently for each cycle. Once this entire JIC-Stage-2 process gets over, the infrequent CGPLN-Labels are purged from the memory.

Algorithm for Jagged Itemset Counting (JIC)-2

1. $Initialize\ all\ the\ variables$
2. $Open\ the\ database\ preDb_n.dat$
3. $Read\ the\ database\ preDb_n$
4. $For\ each\ transaction\ Txn_i\ in\ preDb_n$
5. $numTransactions++$
6. $For\ each\ Item\ TxI_i\ in\ Txn_i$
7. $If\ \varphi(TxI_i) < \sigma_{min}^{G}\ then$
8. $Discard\ TxI_i$
9. $Continue$
10. $Else$
11. $itemSequence \leftarrow TxI_i$
12. $Endif$
13. $Endfor$
14. $TxnCollection \leftarrow sorted\ itemSequence\ based\ on\ its\ support$
15. $headerNodeList \leftarrow itemSequence(FirstItem)$
16. $itemSequence = null$
17. $Endfor$
18. $Close\ the\ database\ preDb_n$
19. $\sigma_{min}^{Local} \leftarrow numTrasactions * threshold$
20. $Build\ Tree\ jTree \leftarrow headerNodeList$
21. $nodeList \leftarrow jNode[]$
22. $For\ each\ Item\ Txn_i\ in\ TxnCollection$
23. $For\ each\ Item\ TxI_i\ in\ Txn_i$
24. $If\ TxI_i\ not\ in\ nodeList\ then$
25. $create\ jNode$

$jNode.data \leftarrow TxI_i$
$jNode.count = 1$
$update\ jNode\ parent\ and\ child\ links$
$nodeList \leftarrow jNode$
$else$
$update\ jNode.count$
$End\ if$
$Endfor$
$jTree \leftarrow nodes\ of\ Txn_i$
$End\ for$
$inFrequentCGPLN \leftarrow inFrequent.tmp\ file$
$discard\ inFrequent.tmp\ file$
$For\ each\ node\ in\ headerNodeList$
$Trace\ route\ using\ parent\ child\ links\ and\ create\ jSet$
$For\ each\ subSet\ \gamma_j\ in\ jSet$
$If\ \varphi(\gamma_j) > \sigma_{min}^{Local}\ then$
$CGPLN - Label_j \leftarrow CGPLN(\gamma_j) \cdot |\gamma_j| \cdot sum(\gamma_j)$
$If\ CGPLN - Label_j\ found\ in\ inFrequentCGPLN$
$inFrequentCGPLN - count \mathrel{+}= \varphi(CGPLN - Label_j)$
$End\ if$
$End\ for$
$jSet = null$
$End\ for$
$Write\ \ CGPLN - Label_j with\ its\ support\ into\ inFrequent.tmp\ file$

E. The Join operation

Now the 'Likely to be frequent CGPLN-Label's from the JIC-Stage-2 process and infrequent CGPLN-Labels from the JIC-Stage-2 are merged with its respective support count and when its count exceeds then GlobalSupportCount σ_min^G, they are moved to the final frequent CGPLN-Labels collection otherwise discarded. Finally the final list of frequently occurring CGPLN-Labels is replaced by its original itemset representation and its support. These itemsets are now added directly to the final Frequent Itemset list and displayed as output.

Algorithm for the Join operation

$Initialize\ all\ the\ variables$
$Open\ the\ database\ inFrequent.tmp$
$inFrequentCGPLN \leftarrow inFrequent.tmp\ file$
$discard\ inFrequent.tmp\ file$
$For\ each\ CGPLN - Label_j\ in\ inFrequentCGPLN$
$If\ CGPLN - Label_j\ found\ in\ CGPN - Labels\ then$
$Update\ CGPLN - Labels[CGPLN - Label_j] - count$
$Else$
$If\ \varphi(CGPLN - Label_j) \geq \sigma_{min}^{G} then$
$CGPN - Labels \leftarrow add\ CGPLN - Label_j$
$Else$
$Discard\ CGPLN - Label_j$
$End\ if$

14. $Endif$
15. $End\ for$
16. $frequentCGPLN \leftarrow frequent.tmp\ file$
17. $discard\ frequent.tmp\ file$
18. $CGPLN - Labels \leftarrow update\ or\ append\ CGPLN - Labels\ from\ frequentCGPLN$
19. $Open\ ReferenceKeys.tmp$
20. $For\ each\ CGPLN - Label_k in\ ReferenceKeys$
21. $if\ CGPLN - Label_k\ found\ in\ CGPLN - Labels\ \ then$
22. $\gamma_k \leftarrow CGPLN - Label_k$
23. $FIS_{nk} \leftarrow add\ \gamma_k$
24. $Endif$
25. $Endfor$
26. $Close\ and\ discard\ ReferenceKeys.tmp$
27. $FIS \leftarrow FIS_{1k}$
28. $FIS \leftarrow FIS_{nk}$
29. $print\ All\ FIS, its\ count\ and\ Execution\ status$

To provide a rigorous mathematical justification and analysis of the superiority of the Jagged Itemset Counting (JIC) algorithm, a combination of formal hypotheses, lemmas, theorems, and proofs is presented. These are based on the properties of frequent itemset mining and complexity analysis, highlighting the advantages of the JIC algorithm compared to other algorithms such as Apriori and Eclat.

Hypothesis:

The JIC algorithm, using Geometric Progression Label Numbers (GPLN) and Cumulative Geometric Progression Label Numbers (CGPLN), outperforms Apriori and Eclat in terms of execution time and memory consumption.

Theorem:

The JIC algorithm achieves a time complexity of $O(n^2)$ and space complexity of $O(n)$, outperforming the Apriori algorithm (with complexity $O(2^n)$ and the Eclat algorithm (with complexity $O(2^n.t)$, where t is the transaction size).

Proof:

Lemma 1: Time Complexity of JIC Algorithm

The JIC algorithm reduces the time complexity by using a geometric progression labeling system (GPLN and CGPLN) and requires only two passes over the database.

Hypothesis:

Each itemset in the transaction database TxDb is labeled using the Geometric Progression Label Number (GPLN). For each itemset γ, the CGPLN label is generated using **Definition-1 (GPLN)** and **Definition-2 (CGPLN).**

Proof:

The process of generating the GPLN and CGPLN requires iterating over each item in F. Since the GPLN computation is a constant-time operation (due to fixed a and r), the total time to generate the CGPLN for a single itemset γ with m items is $O(m)$. Given that we need to process n transactions in the database and generate itemsets for all transactions, the total time complexity becomes:

$$T_{JIC} = O(n\ .m)$$

Assuming m grows linearly with the number of items n, we get:

$$T_{JIC} = O(n^2)$$

This is significantly better than the Apriori algorithm, which needs to generate candidate itemsets of size k, leading to a time complexity of $O(2^n)$.

Lemma 2: Space Complexity of JIC Algorithm
The JIC algorithm requires less space for storing candidate itemsets by using CGPLN labels, which uniquely represent itemsets as numeric values.

Hypothesis:
The space complexity of JIC is driven by the storage of CGPLN labels and the hash maps for counting support values. Since the CGPLN labels for all itemsets can be represented as single numerical values, the space required is proportional to the number of itemsets rather than the size of the itemsets themselves.
Formula:
Let m represent the number of unique items and n the number of transactions. For each itemset, we store:

$$CGPLN(\gamma) = a_1 \,.\, a_2 . a_3$$

where a_1, a_2 and a_3 are the components of the CGPLN label, representing the cumulative value, number of items, and sum of item-values, respectively. The space required to store each CGPLN is $O(1)$.

Proof:
Since the CGPLN labels are stored as fixed-size values, the total space required for all itemsets in the transaction database is proportional to the number of itemsets, denoted as m, which is linear in the size of the database:

$$S_{JIC} = O(n)$$

This is better than Eclat, which stores multiple vertical databases and has a space complexity of $O(2^n \,.\, t)$, where t is the average transaction length.

Lemma 3: Scalability of JIC Algorithm for Big Data
The JIC algorithm scales better for large datasets by partitioning the transaction database and processing each partition independently.

Hypothesis:
The transaction database TxDb is partitioned based on the number of items in each transaction, with frequent itemsets computed in each partition using the CGPLN labels. Let P represent the number of partitions.

Formula:
The total time to process all partitions is given by:

$$T_{JIC-Partitioned} = P \,.\, T_{JIC}$$

Since each partition is processed independently, the complexity remains linear in the number of partitions and quadratic in the number of items:

$$T_{JIC-Partitioned} = O(P \,.\, n^2)$$

Proof:
By dividing the dataset into partitions, JIC minimizes the memory and computation requirements per partition. Each partition only requires processing the transactions within it, and the final frequent itemsets are merged using the Join operation, which has a complexity of $O(n)$. Therefore, the algorithm scales efficiently with large datasets.

So, the JIC algorithm achieves better performance than Apriori and Eclat due to its lower time and space complexities. The use of Geometric Progression Label Numbers (GPLN) and Cumulative Geometric Progression Label Numbers (CGPLN), along with a two-pass scanning approach, enables it to process frequent itemsets in $O(n^2)$ time and $O(n)$ space, compared to $O(2^n)$ for Apriori and $O(2^n . t)$ for Eclat. The algorithm's scalability is also ensured through partitioning, making it suitable for Big Data also.

IV. RESULTS AND EVALUATIONS

The performance of the JIC algorithm is evaluated using both real-world and synthetic databases. The synthetic database is created with IBM Synthetic Data Generator [27] and the real-time databases are downloaded from [26] & [28] and their information is presented in Tables 2 and 3. The dataset downloaded from the website [26] contains 50 thousand rows of Product Information, 3.4 million transaction details, 134 rows of Aisles information, and 21 rows of Department Information. The JIC algorithm's performance is assessed on both artificial and real-world databases. The synthetic database is created with IBM Synthetic Data Generator (https://github.com/zakimjz/ IBMGenerator.git) and the real-time databases are downloaded from [26] & [28] and their information is presented in Tables 2 and 3. 50 thousand rows of product information, 3.4 million transaction details, 134 rows of information about aisles, and 21 rows of department information are all included in the dataset that may be obtained from the website [26]. The CSV (Comma Separated Values) format is used for the dataset.

To continue with data mining, information must be extracted from raw datasets since these data cannot be utilized directly to mine Frequent Itemsets and understanding the buying pattern of sales and the customer's buying behaviour is crucial. A laptop computer with an Intel i5 7200U @ 2.5 GHz - 4 cores and a 2.7 GHz - 1 core CPU, together with 4GB RAM, was used to perform all of the algorithms. This section presents the results of a comparison between the JIC algorithm and two popular algorithms, Apriori and Eclat, regarding the JIC algorithm's efficiency in terms of execution time/runtime and main memory utilization. As illustrated in Fig. 13 and Fig. 15, the Eclat algorithm's resource utilization of heap memory and Central Processing Unit (CPU) consumption is not feasible due to the Online Grocery Database's big size.

A Java Out of capacity Error happens even though all of the algorithms were executed with a 2GB extended heap capacity. Fig. 10 and 11 show how heap memory is used and how much CPU power the JIC algorithm uses during execution, and Fig. 12 and 14 show how heap memory is used and how much CPU power the Apriori algorithm uses during execution using the SPMF software tool [25]. The JIC method performs better than the other algorithms in terms of execution time since it consumes memory uniformly and loads 50% of the CPU during operation.

A. Performance on Execution Time

The JIC method and other algorithms' execution times are displayed in Table 4, and Fig. 2 provides a graphic representation of the same data for the "Online Retail" database. In the

"Online Retail" database, the JIC algorithm performs better than the Apriori and Eclat algorithms for all support thresholds, including the lowest support criterion of 0.1%, as Table 4 and Fig. 2 unambiguously demonstrate. Table 5's "Fruit Hut" database's JIC and other algorithms' execution times are graphically displayed in Fig. 3. According to Table 5 and Fig. 3, the JIC method outperforms the Apriori algorithm and performs comparable to the Eclat algorithm in the "Fruit Hut" real-life database. Perform better, nevertheless, when the support threshold is at 0.1%. For the "T4I4D100K" database in Table 6, the JIC and other algorithms' execution times are visually displayed in Fig. 4. Table 7 lists the JIC and Apriori algorithms' execution timings.

For all support criteria between 0.25 and 5%, the JIC algorithm performs better than the competition. While the JIC algorithm gradually grows with an overall efficiency of 38% when compared to the Apriori algorithm, the Apriori algorithm's execution time climbs constantly as the support threshold drops. The JIC method works quite well for small and medium-sized databases, outperforming both the Apriori and Eclat algorithms in comparable tests. It is concluded that both the Apriori and JIC algorithms find Frequent Itemsets from all sizes of datasets taken for evaluation because the Eclat algorithm was unable to execute for the given Big Data. However, the Eclat algorithm is not suitable when the size of the database or dataset is large, and the JIC algorithm's overall performance on the execution times is good for all databases taken for evaluation.

B. Performance on the usage of Main Memory

Table 8 displays the RAM utilization for the JIC method and other algorithms, while Fig. 6 exhibits the same graphically for the "Online Retail" database. When compared to other methods for this database, it is evident that the JIC approach used less primary RAM. When compared to the other two algorithms, the Apriori and Eclat, the memory utilised by the JIC algorithm during execution in the "Fruit Hut" real-life database and the "T4I4D100K" synthetic database is high (see Tables 9 and 10, Figs. 7 and 8, respectively). However, when the JIC algorithm is conducted on a Big data dataset, Table 11 and Fig. 9 reveal that the primary memory consumed by the JIC algorithm is significantly greater than the Apriori, but no significant difference can be discovered, as shown for small and medium databases or datasets. However, with the higher support threshold, the JIC algorithm consumes less memory than the Apriori approach. Memory occupancy and the support threshold are therefore inversely correlated. At that point, the RAM needed to run the JIC algorithm is increased, and the support threshold is lowered.

TABLE 2

INFORMATION ABOUT THE REAL-LIFE DATABASES USED FOR THE EVALUATION

Database Name	Transaction Count	Item Count	Database Size
Online Retail	5,41,909	2,603	11.4 Mb
Fruit Hut	1,81,970	1,265	3.4 Mb
Insta Cart Online Grocery Shopping Dataset	3.24 Millions (Raw Data) 32,14,873 (Pre-Processed Data)	49,689	551 Mb 179 Mb

TABLE 3

INFORMATION ABOUT THE SYNTHETIC DATABASE – T4I4D100K.DAT USED FOR THE EVALUATION

Particulars of Database	Size
Average Size of Transaction (T)	5
The average size of the Maximal Frequent Itemset (I)	4
Total Number of Items (N)	1000
Total Number of Transactions (D x1000)	1,00,000
Database Size	3.4 Mb

TABLE 4
PERFORMANCE ON EXECUTION TIME FOR "ONLINE RETAIL" DATABASE

Support Threshold %	Apriori Alg. Execution Time(in Sec.)	Execution Time of Eclat Alg. (in Sec.)	Execution Time of JIC Alg. (in Sec.)
0.1	1293	109	22
0.25	424	44	16
0.5	59	22	12
0.75	31	15	8
1	18	11	6

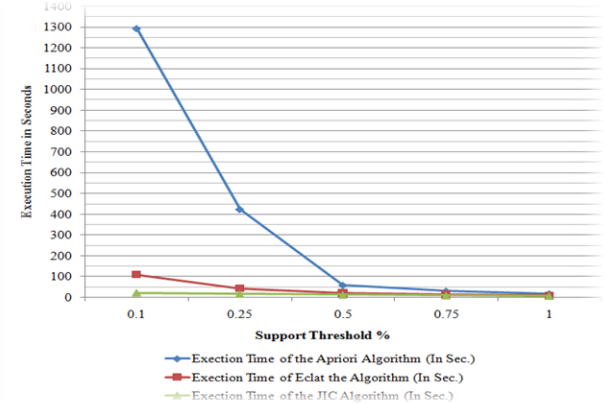

Fig. 2 Performance on Execution Time for "Online Retail" database

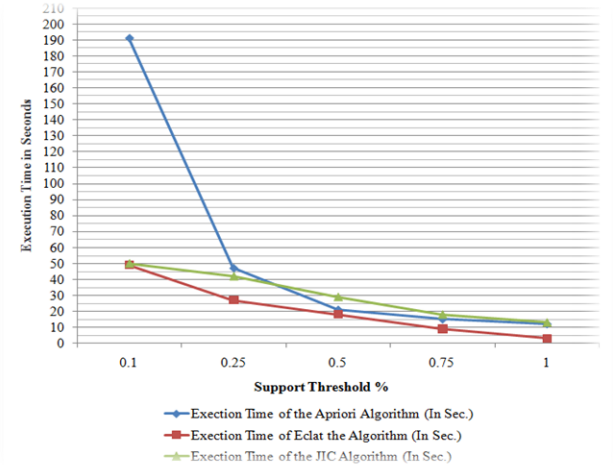

Fig. 3 Performance on Execution Time for "Fruit Hut" database

TABLE 5
PERFORMANCE ON EXECUTION TIME FOR "FRUIT HUT" DATABASE

Support Threshold %	Execution Time of the Apriori Algorithm (in Sec.)	Execution Time of Eclat the Algorithm (in Sec.)	Execution Time of the JIC Algorithm (in Sec.)
0.1	191	49	50
0.25	47	27	42
0.5	21	18	29
0.75	15	9	18
1	12	3	13

TABLE 6
PERFORMANCE ON EXECUTION TIME FOR "T4I4D100K" DATABASE

Support Threshold %	Execution Time of the Apriori Algorithm (in Sec.)	Execution Time of Eclat the Algorithm (in Sec.)	Execution Time of the JIC Algorithm (in Sec.)
0.1	533	63	62
0.25	216	21	52
0.5	50	10	32
0.75	30	5	15
1	6	3	7

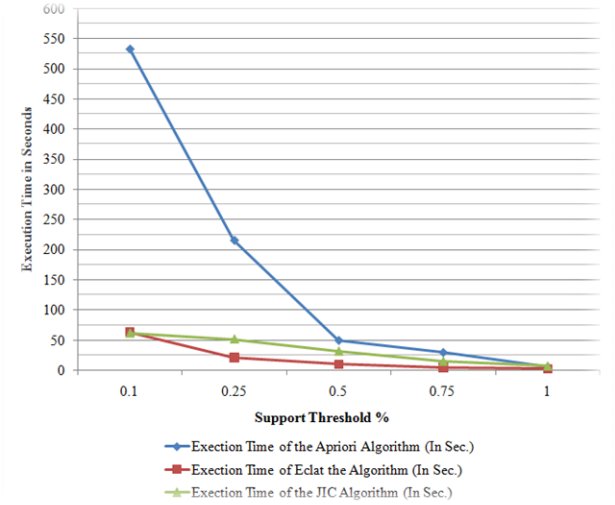

Fig. 4 Performance on Execution Time for "T4I4D100K" database

TABLE 7
PERFORMANCE ON EXECUTION TIME FOR "INSTA CART ONLINE RETAIL" DATABASE

Support Threshold %	Execution Time of Apriori Alg.(in Sec.)	Execution Time of the JIC Alg. (in Sec.)
0.25	1682	1065
0.5	554	290
0.75	213	140
1	134	83
2	40	23
5	25	13

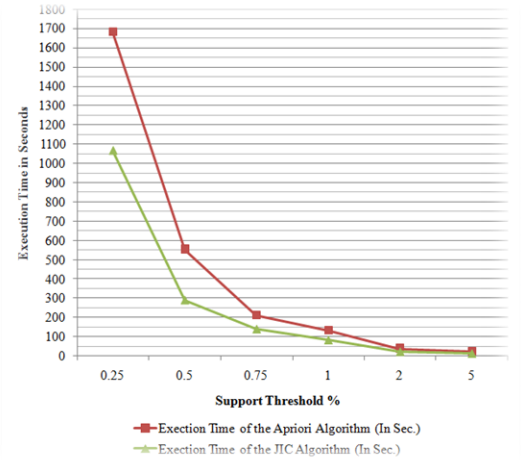

Fig. 5 Performance on Execution Time for "Insta Cart Online Retail" database

TABLE 8

PERFORMANCE ON MEMORY CONSUMPTION FOR "ONLINE RETAIL" DATABASE

Support Threshold %	Memory Used by the Apriori Algorithm (in MB)	Memory Used by the Eclat Algorithm (in MB)	Memory Used by the JICAlgorithm (in MB)
0.1	316	289	143
0.25	317	263	153
0.5	307	250	63
0.75	240	238	86
1	241	195	76

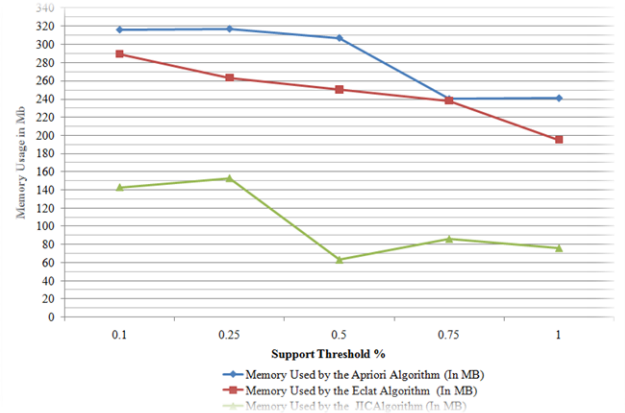

Fig. 6 Performance on Memory Consumption for "Online Retail" database

TABLE 9

PERFORMANCE ON MEMORY CONSUMPTION FOR "FRUIT HUT" DATABASE

Support Threshold %	Memory Used by the Apriori Algorithm (in MB)	Memory Used by the Eclat Algorithm (in MB)	Memory Used by the JICAlgorithm (in MB)
0.1	123	286	546
0.25	116	220	470
0.5	110	206	431

0.75	70	196	202
1	67	110	162

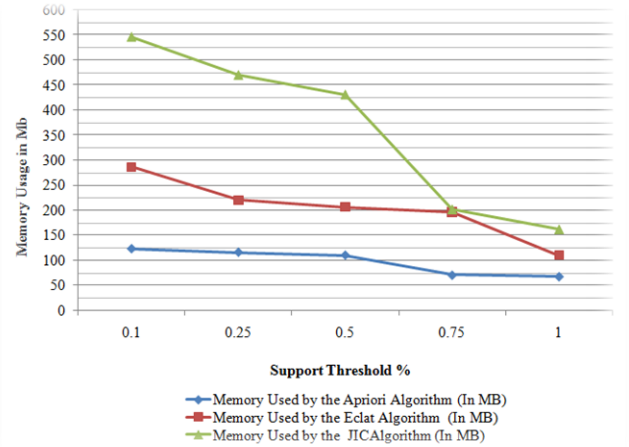

Fig. 7 Performance on Memory Consumption for "Fruit Hut" database

TABLE 10

PERFORMANCE ON MEMORY CONSUMPTION FOR "T4I4D100K" DATABASE

Support Threshold %	Memory Used by the Apriori Alg(in MB)	Memory Used by the Eclat Alg (in MB)	Memory Used by the JIC Algorithm (in MB)
0.1	302	130	469
0.25	126	105	370
0.5	121	87	262
0.75	61	87	270
1	61	50	227

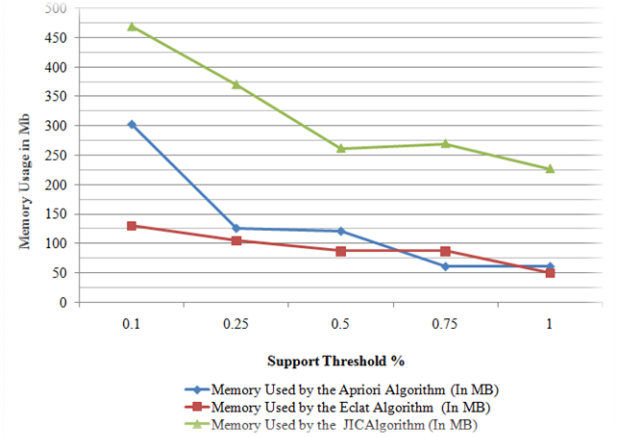

Fig. 8 Performance on Memory Consumption for "T4I4D100K" database

TABLE 11

PERFORMANCE ON MEMORY CONSUMPTION FOR "INSTA CART ONLINE RETAIL" DATABASE

Support Threshold %	Memory Used by the Apriori Algorithm (in MB)	Memory Used by the JICAlgorithm (in MB)

0.25	804	1352
0.5	640	1295
0.75	635	988
1	602	708
2	579	705
5	578	483

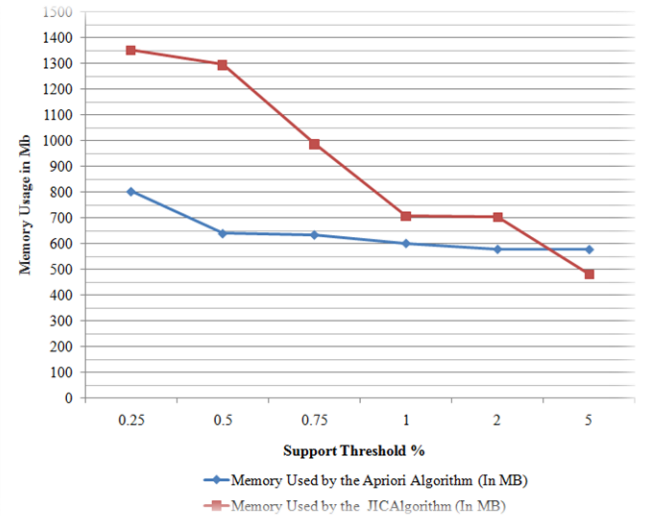

Fig. 9 Performance on Memory Consumption for "Insta Cart Online Retail" database

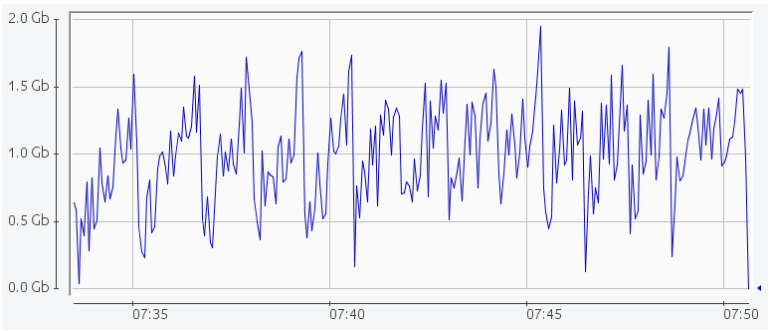

Fig. 10 Graph taken from the Java Monitoring and Management Console – Heap Memory Usage during the execution of the JIC algorithm Execution process

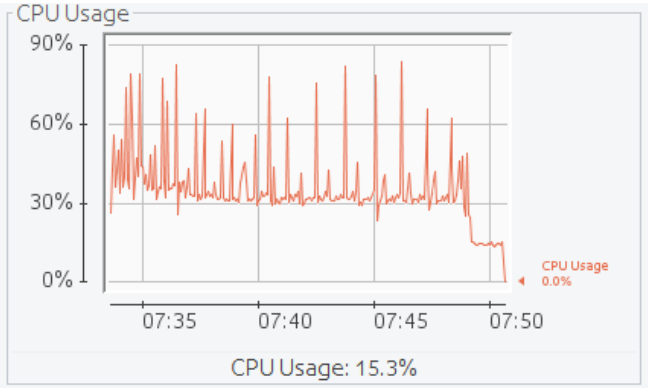

Fig. 11 Graph taken from the Java Monitoring and Management Console – CPU Usage during the execution of the JIC algorithm Execution process

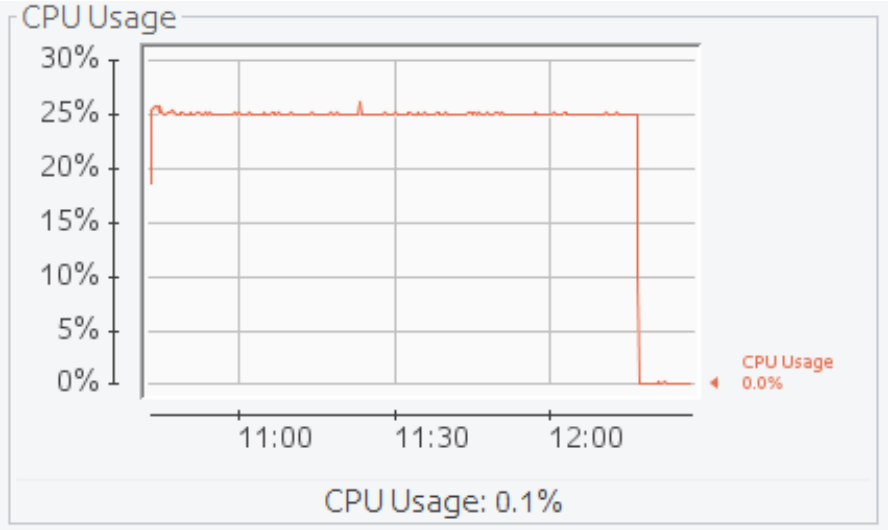

Fig. 12 Graph taken from the Java Monitoring and Management Console – CPU Usage during the execution of the Apriori algorithm using the SPMF software tool

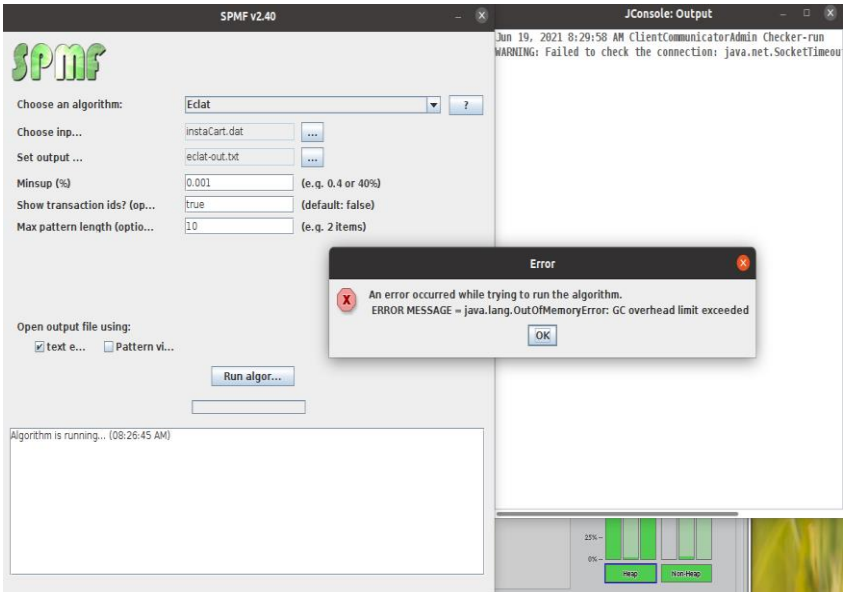

Fig. 13 Failure-Screenshot of the Eclat Algorithm during execution using the SPMF software tool

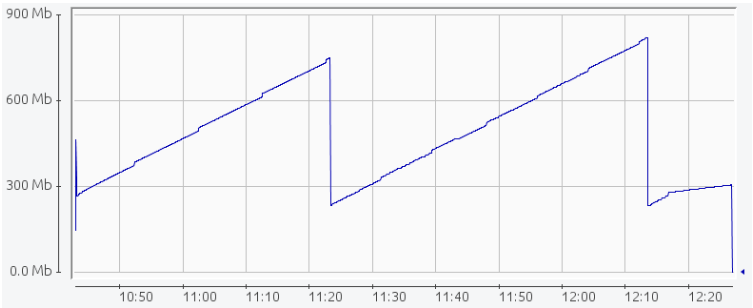

Fig. 14 Graph taken from the Java Monitoring and Management Console – Heap Memory Usage during the execution of the Apriori algorithm using the SPMF software tool

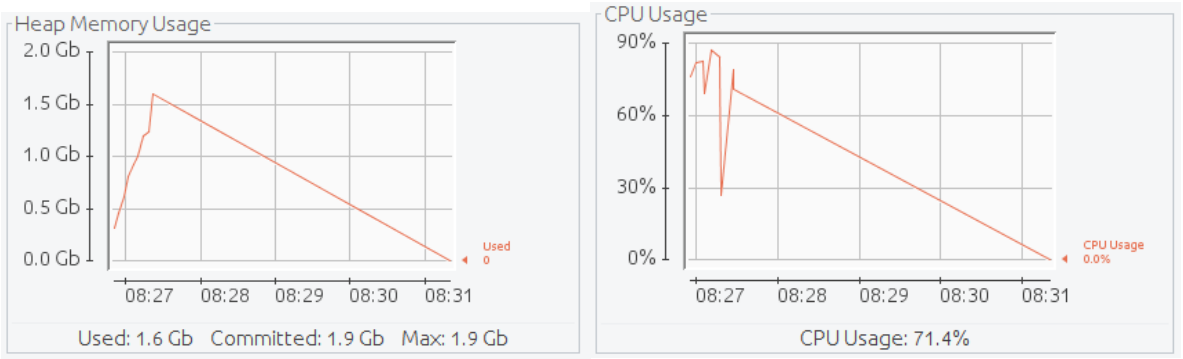

Fig. 15 Graph taken from the Java Monitoring and Management Console – Heap Memory Usage and CPU Usage during the execution of the Eclat algorithm using SPMF

V. CONCLUSION AND FUTURE DIRECTION OF RESEARCH

In the process of Knowledge Discovery from Data (KDD), the analysis of Frequent Itemset Mining (FIM) produces very useful information such as how the items or data are correlated with each other and how frequently it is found in the database, particularly in transaction data and its report is critical for decision-makers to procure and stock items. As the stock is the company's asset and the company's profit is dependent on the asset's liquidity, Frequent Itemset mining was given more importance. Finding Frequent Item Sets is a crucial and time-consuming task in the field of data mining. The variety and volume of data are increasing dramatically in the current Big Data era, making it difficult to mine valuable information in a timely manner. Despite the fact that many researchers proposed many different types of algorithms for Frequent Item Set mining, they are not capable of finding the frequent itemsets using the normal computing facility from the real-life big-sized dataset particularly when the support count was low and they could not perform efficiently when the dataset was big.

This paper presented an innovative approach to overcoming this difficulty and gaining new insights from data sources. Instead of a Complex Data Structure, all itemsets, whether frequent or infrequent, will be stored in the form of a simple label structure. Hashing technique is used to store the CGPLN-Labels and Frequent Itemsets and used temporary files to store the CGPLN-Label representations to reduce the use of huge volume of main memory. As a result, the amount of memory used is uniformly reduced throughout the algorithm implementation. This paper presents the Jagged Itemset Counting algorithms. Despite the prevalence of itemset mining algorithms in the literature, the JIC algorithm will read the database twice, regardless of its size or the number of transactions, without transforming the database layout from horizontal to vertical or constructing any conditional databases. When compared to the Apriori and Eclat algorithms, all frequent Itemsets up to size k can be found from the database in two passes.

When compared to the Apriori and Eclat algorithms, the JIC algorithm's use of compact and unique label representation with a simple counting mechanism outperforms the Apriori and Eclat algorithms for small and medium data and shows better performance in execution time even at the lowest support threshold. The Frequent Itemsets identified by this proposed JIC algorithm consistent with the results of state-of-the-art methods. In the case of Big Data, while the FP-Growth and Eclat algorithms failed to perform, the proposed technique performed better in terms of execution time, main memory consumption and disc memory utilization.

In the future, the JIC algorithm will be improved to reduce the number of subsets and implementing the JIC algorithm in parallel or distributed computing will almost certainly result in greater efficiency in all aspects. The JIC algorithm can also be further modified to mine the

frequent itemsets from the data-streams. The JIC algorithm can be further modified to mine the high-utility itemsets, top-k-rank itemsets, maximal-frequent itemsets, closed-frequent itemsets and fault-tolerant itemsets.

REFERENCES

[1] Ilamchezhian, J & Cyril Raj, V 2021, 'A Novel Approach For Frequent Itemset Mining Using Geometric Progression Number Labeling', Turkish Journal of Computer and Mathematics Education, vol. 12, no. 10, pp. 3529–3538

[2] Ilamchezhian, J., Raj, V.C. and Kannan, A., 2021. Jagged Itemset Counting for Mining Frequent Itemsets. Design Engineering, pp.1144-1161.

[3] Wu, X, Zhu, X, Gong–Qing Wu & Ding, W 2014, 'Data mining with big data', IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 1, pp. 97–107

[4] Agrawal, R, Imielinski, T & Swami, AN 1993, In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (ICDM '93), May 26–28, Mining association rules between sets of items in large databases, Washington, DC, USA, pp. 207–216

[5] Agrawal, R & Srikant, R 1994, In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94), September 12 – 15, Fast algorithms for mining association rules, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 487–499

[6] Zaki, MJ 2000, 'Scalable Algorithms for Association Mining', IEEE Transactions on Knowledge and Data Engineering, vol. 12, no. 3, pp. 372–390

[7] Jiawei Han, Jian Pei, Yiwen Yin & Runying Mao 2004, 'Mining frequent patterns without candidate generation A frequent–pattern tree approach', Data Mining and Knowledge Discovery, vol. 8, pp. 53–87

[8] Ashok Savasere, Edward Omiecinski & Shamkant Navathe 1995, In Proceedings of 21st International Conference on Very Large Data Bases, September 11 – 15, An Efficient Algorithm for Mining Association Rules in Large Databases, Zurich, Switzerland, pp. 432–444

[9] Jong Soo Park, Ming–Syan Chen & Philip S.Yu 1995, In Proceedings of the SIGMOD '95 ACM SIGMOD International Conference on Management of data, May 22–25, An Effective Hash Based Algorithm for Mining Association Rules, San Jose California USA, vol. 24, no. 2, pp. 175–186

[10] Brin, S, Motwani, R, Ullman, JD & Tsur, S 1997, In Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, May 11 – 15, Dynamic itemset counting and implication rules for market basket data, Tucson, Arizona, USA, pp. 255–264

[11] Veloso, A, Otey, ME, Parthasarathy, S & Meira, W 2003, In Proceedings of 10th International Conference on High–Performance Computing(HiPC 2003), December 17–20, Parallel and Distributed Frequent Itemset Mining on Dynamic Datasets, Springer, Berlin, Heidelberg, vol. 2913

[12] Goethals, B 2003, 'Survey on frequent pattern mining', University of Helsinki Journal, vol. 19, pp. 840–852

[13] Sivanandam, SN, Sumathi, S, Hamsapriya, T & Babu, K 2004, 'Parallel Buddy Prima – A Hybrid Parallel Frequent itemset mining algorithm for very large databases', Academic Open Internet Journal, vol. 13, Corpus ID: 14102089

[14] Yun, U & Leggett, J 2005, In Proceedings of the SIAM International Conference on Data Mining (SDM 2005), April 21–23, WFIM: Weighted frequent itemset mining with a weight range and a minimum weight, Newport Beach, CA, USA

[15] Gouda, K, Hassaan, M & Zaki, MJ 2010, 'Prism: An effective approach for frequent sequence mining via prime–block encoding', Journal of Computer and System Sciences, vol. 76, no. 1, pp. 88–102
[16] Yu, X & Wang, H 2014, 'Improvement of Eclat Algorithm Based on Support in Frequent Itemset Mining', Journal of Computers, vol. 9, no. 9, pp. 2116–2123
[17] Cagliero, L & Garza, P 2014, 'Infrequent weighted itemset mining using frequent pattern growth', IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 4, pp. 903–915
[18] Essalmi, H Far, M, El, Mohajir, M, El & Chahhou, M 2016, In Proceedings of 2016 4th IEEE International Colloquium on Information Science and Technology (CiSt), October 24–26, A novel approach for mining frequent itemsets: AprioriMin, Institute of Electrical and Electronics Engineers (IEEE), Tangier, Morocco, pp. 286–289
[19] Vo, B, Le, T, Coenen, F & Hong, TP 2016, 'Mining frequent itemsets using the N–list and subsume concepts', International Journal of Machine Learning and Cybernetics, vol. 7, no. 2, pp. 253–265
[20] Zhang, C, Zhang, X & Tian, P 2017, In Proceedings of IEEE 2nd International Conference on Data Science in Cyberspace, June 26–29, An Approximate Approach to Frequent Itemset Mining, Institute of Electrical and Electronics Engineers (IEEE), Shenzhen, China. pp. 68–73
[21] Klangwisan, K & Amphawan, K 2017, In Proceedings of 2017 9th International Conference on Knowledge and Smart Technology (KST), February 1–4, Mining weighted–frequent–regular itemsets from transactional database, ECTI, Pattaya, Thailand, pp. 66–71
[22] Djenouri, Y, Djenouri, D, Belhadi, A & Cano, A 2018, 'Exploiting GPU and cluster parallelism in single scan frequent itemset mining. Information Sciences', Information Sciences, vol. 496, pp. 363–377
[23] Yasir, M, Habib, MA, Ashraf, M, Sarwar, S, Chaudhry, MU, Shahwani et al. 2019, 'TRICE: Mining Frequent Itemsets by Iterative TRimmed Transaction LattICE in Sparse Big Data', IEEE Access, vol. 7, pp. 181688–181705
[24] Huang, PY, Cheng, WS, Chen, JC, Chung, WY, Chen, YL & Lin, KW 2021, 'A Distributed Method for Fast Mining Frequent Patterns From Big Data', IEEE Access, vol. 9, pp. 135144–135159
[25] Fournier–Viger, P, Lin, JCW, Dinh, T & Le, HB 2016, In Proceedings of 11th International Conference on Hybrid Artificial Intelligent, April 18–20, Mining Correlated High–Utility Itemsets using the Bond Measure, Hybrid Artificial Intelligence Systems, Seville, Spain, vol. 18, no. 20, pp. 53–65
[26] The Instacart Online Grocery Shopping Dataset 2017, Accessed on <24–06–2019> from https://www.instacart.com/datasets/grocery–shopping–2017
[27] https://github.com/zakimjz/IBMGenerator.git
[28] http://www.philippe–fournier–viger.com/spmf/datasets

Dr. Ilamchezhian J is a distinguished academic and researcher in the field of Computer Science and Engineering. He holds a Ph.D. and M.Tech. in Computer Science and Engineering from Dr. M.G.R. Educational and Research Institute, an M.Sc. in Information Technology from the University of Madras, an M.B.A. in HR and Systems from Anna University, and an M.Phil. in HR from Vinayaka Missions University. He has an impressive publication record, with 7 papers in Scopus Indexed Journals, 10 papers in National Conferences, and 8 papers in International Conferences. His research interests encompass Big

Data Analytics, Parallel Computing, Grid Computing, Cloud Computing, and Data Mining. Since July 2014, he has been serving as an Associate Professor at Dr. M.G.R. Educational and Research Institute, Maduravoyal, Chennai. His dedication to teaching and research, combined with his extensive industry and academic experience, has established him as a highly respected figure in the field of Computer Science and Engineering.

Dr. Kannan A. is a retired Professor from the Department of Information Science and Technology at Anna University, India. With a distinguished career in both academia and industry, he has made significant contributions to the fields of Network Security, Data Mining, Artificial Intelligence, and Software Engineering. His professional journey includes experience as a Computer Programmer at Bhabha Atomic Research Centre, Mumbai. His extensive research has led to numerous publications and books in his areas of expertise. He actively serves as an editorial member and reviewer for several international reputed journals, reflecting his commitment to advancing knowledge and maintaining high standards in scientific publishing. In addition to his research and academic roles, he has successfully completed various administrative responsibilities and is a member of numerous international affiliations.

Prof. Dr. Vellankanni Cyril Raj is a distinguished academic and researcher with extensive expertise in Computer Science and Engineering. He was a Professor of Computer Science and Engineering/Information Technology and former Dean of Engineering and Technology and at Dr. M.G.R. Educational and Research Institute University, Chennai. He earned his Ph.D. from Jadavpur University, Kolkata, in 2009, his M.E. from the Government College of Technology, Coimbatore, in 1995, and his B.E. from PSNA College of Engineering & Technology in 1988. His academic career spans over three decades, during which he has held significant positions, including Dean, Additional Dean and Head of Department at various institutions. He has supervised numerous Ph.D. candidates, producing impactful research and contributing to the academic growth of several institutions in South India. His dedication to academia and research continues to inspire and influence the field of Computer Science and Engineering. He has supervised numerous Ph.D. candidates, produced impactful research and contributed to the academic growth of several institutions in South India. His dedication to academia and research continues to inspire and influence the field of Computer Science and Engineering.

N. Padmapriya completed her M.Sc. (five-year integrated programme) in statistics at the Pondicherry University in 2013. She qualified CSIR-NET in the year 2017. Currently, she is working as Assistant Professor in the Department of Statistics, Sri Sarada College for Women (Autonomous), Salem-16 and pursuing her Ph.D. in the Department of Statistics, Pondicherry University.