Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 04, April-2025

Bisect and Rule Accord Algorithm (BIRA) to Discover Frequent Itemset

Harshitha.P, Saheena Begam.A

ABSTRACT
Mining large datasets to extract meaningful hidden patterns has long been an area of research, but existing methods still face significant challenges related to computational cost, time, and memory overheads. This paper addresses these challenges by introducing a novel approach that bisects large datasets into smaller partitions. For each partition, the Bisected Count of elements is calculated, which measures the frequency of patterns within that subset of the divided data. After processing the bisected data, the Combined Count of the elements is found by combining the support counts from each bisection, resulting in a final, comprehensive support count for the entire dataset. To efficiently perform these calculations, the algorithm leverages a bit vector approach, which simplifies the process of determining pattern frequencies while reducing memory, time complexity and helps manage large datasets without overwhelming system resources. The proposed algorithm BIRA is evaluated against many states of the art existing algorithms, with a focus on two critical performance metrics: speed and memory consumption. Experimental results demonstrate that the proposed algorithm significantly reduces both time and memory overheads when compared to traditional approaches, making it a promising solution for large-scale data mining tasks.

Keywords
BIRA algorithm, big data, frequent itemset, data mining, minimum support count, bisection approach, FIM

INTRODUCTION

One of the fundamental tasks in data mining, especially in the area of association rule learning, is frequent itemset mining (FIM). It entails locating collections of items (or itemsets) that commonly occur together in a dataset. These frequent itemsets are crucial for identifying patterns and correlations in data [1], and they may be applied to a variety of tasks, including fraud detection, recommender systems, and market basket analysis.

Concepts in Frequent Itemset Mining

1. Itemset
   An itemset is a collection of one or more items from a dataset. For example, in a market basket context, if a dataset contains transactions from a grocery store, an itemset could be {bread, milk, butter}.

2. Support Count
   Support is a measure that indicates how often an itemset appears in the dataset. Formally, it is the proportion of transactions that contain the itemset. The support of an itemset is important because it helps identify which itemsets are frequent.

Example: If {A, B} appears in 60 out of 100 transactions, its support is 60%.

3. Frequent Itemset

An itemset is considered "frequent" if its support is greater than or equal to a user-defined minimum support threshold. This threshold determines the frequency level needed for an itemset to be considered useful.

Example: If the minimum support threshold is set at 50%, then an itemset must appear in at least 50% of transactions to be deemed frequent.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 04, April-2025

ALGORITHMS FOR FREQUENT ITEMSET MINING

1. Apriori Algorithm
   The Apriori algorithm [1] is one of the earliest and most well-known algorithms for mining frequent itemsets. It works by identifying frequent individual items in the dataset and expanding them to larger itemsets as long as those larger itemsets have sufficient support. It uses a "bottom-up" approach and prunes the search space based on the observation that any subset of a frequent itemset must also be frequent.
2. SPC – Single pass Count
   The Single Pass Counting (SPC) algorithm [2] is a parallelized version of the Apriori algorithm, utilizing the MapReduce framework to optimize performance. In this implementation, the process of counting the support for candidate itemsets is distributed across multiple nodes, enabling faster computation by leveraging parallel processing. The entire SPC algorithm is divided into two distinct phases, which allows it to efficiently manage the data and operations. By adopting this approach, the SPC algorithm successfully mitigates several of the inefficiencies and limitations inherent in the traditional Apriori algorithm, such as its high computational cost and multiple database scans.
3. ECLAT (Equivalence Class Transformation)
   ECLAT is another frequent itemset mining algorithm that uses a depth-first search approach and organizes data into vertical itemset representations, improving performance by avoiding the need to repeatedly scan the dataset.
4. parECLAT (Parallel Equivalence Class Transformation)
   This is an extension of the ECLAT algorithm that leverages parallel processing to improve performance when mining frequent itemsets from large datasets. ECLAT, the base algorithm, works by transforming the dataset into a vertical format and using a depth-first search strategy to efficiently find frequent itemsets. However, as datasets grow larger, the original ECLAT algorithm can become slow and resource-intensive. parECLAT [3] aims to solve this issue by distributing the workload across multiple processors or machines.

PROBLEM STATEMENT

The main problem while discovering the frequent itemset is scanning the input database repeatedly many times [5]. The problem of scanning the input database multiple times to discover frequent itemsets indeed leads to significant computational overhead [4]. Frequent itemset mining often involves multiple passes over the database [6], consuming both time and resources. Popular algorithms like Apriori and Eclat aim to minimize these scans, but they still require multiple passes depending on the dataset's size and structure. While many algorithms today aim to reduce the number of database scans, the proposed method goes a step further by not even requiring a full scan of the database. Instead, it processes and retrieves the necessary elements by scanning only half of the database, leading to increased efficiency.

PROPOSED APPROACH

The proposed approach optimizes data retrieval by implementing a bisecting mechanism that drastically minimizes the number of database scans or iterations. Rather than scanning the entire database, it begins by checking if the input file has not yet reached the end (EOF). It then retrieves the first row while simultaneously accessing the EOF row.
Next, a comparison is made between the upper bound of the data and the value of EOF minus one. If this condition is met, it indicates that the midpoint of the dataset has been reached. At this point, the loop terminates, ensuring that only half of the database is scanned.
This technique effectively reduces processing time and computational resources by leveraging the bisection process. Instead of performing repetitive scans across the entire database, the approach narrows down the data to the necessary portion, halting once the middle of the dataset is reached. By cutting the number of iterations in half, it significantly enhances efficiency, making it particularly useful for large datasets where frequent scanning is resource-intensive. In summary, this mechanism focuses on optimizing retrieval by reducing full database scans, leveraging comparisons between the initial row and EOF, and terminating early when the midpoint is reached. The process diagram is shown in the figure 1.

Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 04, April-2025

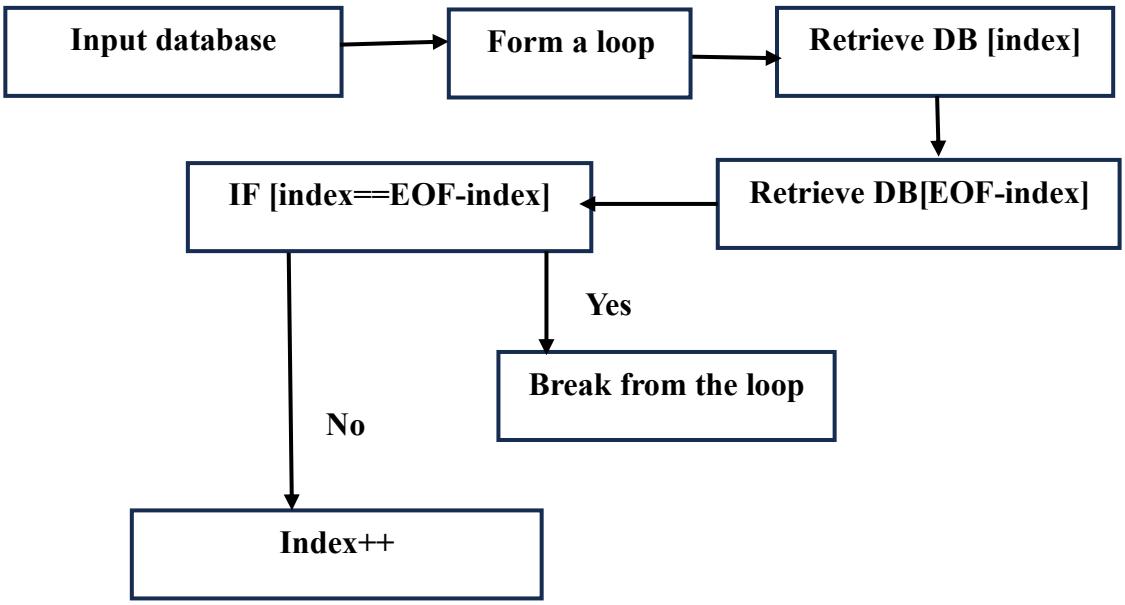

Figure 1: Process diagram of the BisectData

```
Procedure bisectDataRetrieval(database Dd)
start = 0
   end = getEOF(Dd)
     IF end < 0:
      RETURN "Database is empty"
     WHILE [start < end]
      UpperHalf = database[start]
      LowerHalf = database[end]
            IF [start ==(end-start)]
            Break
            ELSE:
            start = start +1
            Close IF
    Close While
End Procedure
```

Figure 2: Pseudo code to bisect the database

EXPLANATION

1. Initialization: The function initializes two pointers: start (first row) and end (EOF, last row).
2. Empty Check: It checks if the database is empty and returns a message if it is.
3. Bisecting Process: The loop runs while the start pointer is less than the end pointer.
4. Midpoint Calculation: At each iteration, it calculates the midpoint of the dataset and compares the values at the start and end pointers to see if the midpoint is reached.
5. Pointer Adjustment: Depending on the comparison, the algorithm adjusts the start or end pointer, reducing the search space.
6. Termination: The loop terminates once the midpoint is reached or the search is narrowed down to a small portion of the data, effectively cutting the number of scans in half.

The sample input database is shown in the following table 1, which is processed to get the retrieved result both the upper half and the lower half of the data separately showcased in the table 2.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 04, April-2025

TID	ITEMS
T1	R1, R2, R3, R5, R6, R15
T2	R1, R3, R7
T3	R5, R9
T4	R1, R3, R4, R5, R7
T5	R1, R3, R5, R7, R12
T6	R5, R10
T7	R1, R2, R3, R5, R6, R16
T8	R1, R3, R4
T9	R1, R3, R5, R7, R13
T10	R1, R3, R5, R7, R14

Table 1: Sample input database

UPPER HALF			LOWER HALF	
TID	ITEMS		TID	ITEMS
T1	R1, R2, R3, R5, R6, R15		T1	R1, R3, R5, R7, R14
T2	R1, R3, R7		T2	R1, R3, R5, R7, R13
T3	R5, R9		T3	R1, R3, R4
T4	R1, R3, R4, R5, R7		T4	R1, R2, R3, R5, R6, R16
T5	R1, R3, R5, R7, R12		T5	R5, R10

Table 2: Bisected input database

Database Structure: There are 10 rows of data, and we are analysing distinct items found in these rows.

Bisected Count (BC): This term refers to the count of distinct items that is split between two parts of the database:

- **Upper part**: Represents one part of the database (could be the first half).
- **Lower part**: Represents the other part (second half).

The Bisected Count (BC) is the count of items in each of these parts.

User-defined Support: This is a threshold value that is provided by the user. In this case, the support value is set to **4**. This value will help in pruning items with low frequency (i.e., less than 4 occurrences).

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 04, April-2025

Combined Count (CC): After the BCs for both the upper and lower parts of the database are computed, the **Combined Count** (CC) is calculated by summing the BC from the upper and lower parts for each item. This gives the overall frequency of each distinct item in the entire database.

Pruning: Once the CC is computed, any items whose combined count is less than the user-defined support (in this case, 4) are **pruned**. This means they are removed from consideration as they do not meet the minimum required frequency threshold.

The resultant of the distinct items is shown in the following table 3.

UPPER HALF		LOWER HALF	
ITEMS	**BC**	**ITEMS**	**BC**
R1	4	**R1**	4
R2	1	**R2**	1
R3	4	**R3**	4
R4	1	**R4**	1
R5	4	**R5**	4
R6	1	**R6**	1
R7	2	**R7**	3
R9	1	**R10**	1
R12	1	**R13**	1
R15	1	**R14**	1`
		R16	1

Table 3: Bisected count found for upper and lower parts

Except four elements R1, R3, R5, and R7 all other items are pruned as the combined count CC values is lesser than the user defined support count value 5. The final items retained is shown in the table 4.

DISTINCT ITEM	CC
R1	8
R3	8
R5	8
R7	5

Table 4: Final distinct items retained after pruning

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 04, April-2025

PROPERTY 1

"If an element's support count (how many times the element appears) is less than this user-specified threshold, it is classified as an infrequent element. These infrequent elements are usually pruned (i.e., removed) from further analysis because they don't meet the criteria of being sufficiently frequent in the dataset." [7]

UPPER PART			LOWER PART	
TID	**ITEMS**		**TID**	**ITEMS**
T1	R1, R3, R5		**T1**	R1, R3, R5, R7
T2	R1, R3, R7		**T2**	R1, R3, R5, R7
T3	R5		**T3**	R1, R3
T4	R1, R3, R5, R7		**T4**	R1, R3, R5
T5	R1, R3, R5, R7		**T5**	R5

Table 5: Final items in the database

The next step, as described, involves creating a **bit vector table** to represent the pruned database. A bit vector table is a binary representation of the transactional data, where each element (item) is represented by either a 1 or 0:

- **1** indicates that the element (item) is **present** in the corresponding transaction (row).
- **0** indicates that the element is **not present** in the transaction.

Table 4: Binary presentation of the bisected database

UPPER PART						LOWER PART					
ITEMS	**T1**	**T2**	**T3**	**T4**	**T5**	**ITEMS**	**T1**	**T2**	**T3**	**T4**	**T5**
R1	1	1	0	1	1	**R1**	1	1	1	1	0
R3	1	1	0	1	1	**R3**	1	1	1	1	0
R5	1	0	1	1	1	**R5**	1	1	0	1	1
R7	0	1	0	1	1	**R7**	1	1	0	0	0

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 04, April-2025

PROPERTY 2

"An itemset may not meet the minimum support threshold in one particular partition (i.e., its support count is too low in that part of the dataset). This would cause it to be marked as infrequent locally within that partition. When the entire dataset (all partitions) is merged, the combined support count of the itemset is calculated by summing its counts across all partitions. This may result in the itemset meeting or exceeding the global minimum support threshold, making it a frequent itemset in the merged dataset, even if it was infrequent in individual partitions." [8]

BIRA ALGORITHM

Bisection of the Dataset

- The dataset is split into two parts: the **upper part** and the **lower part**.
- This is helpful when dealing with large datasets, as it allows parallel computation and easier management of data.

Bisected Count (BC)

- The bisected count (BC) is computed individually for both the upper and lower partitions of the dataset.
- The BC for an item or itemset represents how often the item appears in that particular partition.
- The same operation is performed for both parts to obtain counts.

Combined Count (CC)

- After computing the BC for both partitions, the combined count (CC) is calculated by summing the counts from the upper and lower partitions.
- The CC represents the total occurrences of an item or itemset in the entire dataset (i.e., across both partitions).

Comparison with Minimum Support

- The user-defined minimum support threshold is used to filter itemsets.
- If the combined count (CC) of an item or itemset is greater than or equal to the minimum support, the item or itemset is considered frequent and is retained for further consideration.
- If the CC is less than the minimum support, the item or itemset is pruned (i.e., discarded) as it is not frequent enough.

Forming the Next (n+1) Itemsets

- If an itemset meets the minimum support threshold, it is used to form the next level of itemsets (i.e., n+1 itemsets), where n is the number of items in the current itemset.
- For example, if the current level has 1-itemsets (individual items), and the itemsets pass the support threshold, then the algorithm will attempt to form 2-itemsets (combinations of two items).
- The algorithm continues forming larger itemsets (n+1) until no more frequent itemsets can be found, or all possible combinations are generated.

Binary Operations

- The binary operation mentioned likely refers to the bit vector representation you mentioned earlier.
- For example, to check the presence of an item in both the upper and lower partitions, the binary values (1 for presence, 0 for absence) can be combined using AND operation.
- The combined bit vectors for itemsets can be used to efficiently calculate support counts.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 04, April-2025

ALGORITHM BIRA (DATABASE Dd, SUPPORT Ss)

Input: Dd (dataset), Support Ss
Output: Frequent itemsets

```
1. Split Dd into two partitions: D_upper and D_lower
2. Initialize frequent_itemsets = []
3. For each item i in D:
    Compute BC_upper[i] = count of i in D_upper
    Compute BC_lower[i] = count of i in D_lower
4. For each item i:
    CC[i] = BC_upper[i] + BC_lower[i]
        If CC[i] >= minsup:
       frequent_itemsets.append({i})
    Else:
       prune item i
5. Set k = 1
6. While frequent_itemsets of size k is not empty:
    Generate candidate itemsets of size (k + 1) from frequent_itemsets of size k
        For each candidate itemset c:
       Compute BC_upper[c] = count of c in D_upper
       Compute BC_lower[c] = count of c in D_lower
       CC[c] = BC_upper[c] + BC_lower[c]
       If CC[c] >= minsup:
          Add c to frequent_itemsets for (k + 1)
       Else:
          prune itemset c
        Increment k by 1

    7.  Output all frequent_itemsets
End algorithm
```

Figure 3: Pseudo code of the BIRA algorithm

EXPLANATION OF BIRA

The elements that are retained are permuted with the simple binary AND operation to find the bisected count of the both parts as shown in the following section,

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 04, April-2025

Initially consider R1 and R3

UPPER PART							STATUS
ITEM	**T1**	**T2**	**T3**	**T4**	**T5**	**BC**	
R1	1	1	0	1	1		
R3	1	1	0	1	1	&	
R1, R3	1	1	0	1	1	4	
LOWER PART							
R1	1	1	1	1	0		
R3	1	1	1	1	0	&	
R1, R3	1	1	1	1	0	4	
COMBINED COUNT CC = BC + BC						**8**	**RETAINED**
UPPER PART							
R1, R3	1	1	0	1	1		
R5	1	0	1	1	1	&	
R1, R3, R5	1	0	0	1	1	3	
LOWER PART							
R1, R3	1	1	1	1	0		
R5	1	1	0	1	1		
R1, R3, R5	1	1	0	1	0	3	
COMBINED COUNT CC = BC + BC						**6**	**RETAINED**
UPPER PART							
R1, R3, R5	1	0	0	1	1		
R7	0	1	0	1	1	&	
R1, R3, R5, R7	0	0	0	1	1	2	
LOWER PART							
R1, R3, R5	1	1	0	1	0		
R7	1	1	0	0	0	&	
R1, R3, R5, R7	1	1	0	0	0	2	
COMBINED COUNT CC = BC + BC						**4**	**PRUNED**

The itemsets {R1, R3} and {R1, R3, R5} and are frequent since its counts are higher than the user defined support count values. Similarly, for the next permutation R1 and R5 are considered and the calculations are carried out as shown in the next section. The other items are fetched to compute the bisected count and then the combined count to check whether it can be retained to do the n+1 computation or just pruned away without continuing the n+1 computation which reduces the overall execution time and saves the overhead cost of computations.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 04, April-2025

UPPER PART							STATUS
ITEM	**T1**	**T2**	**T3**	**T4**	**T5**	**BC**	
R1	1	1	0	1	1		
R5	1	0	1	1	1	&	
R1, R5	1	0	0	1	1	3	
LOWER PART							
R1	1	1	1	1	0		
R5	1	1	0	1	1	&	
R1, R5	1	1	0	1	0	3	
COMBINED COUNT CC						**6**	**RETAINED**
UPPER PART							
R1, R5	1	0	0	1	1		
R7	0	1	0	1	1	&	
R1, R5, R7	0	0	0	1	1	2	
LOWER PART							
R1, R5	1	1	0	1	0		
R7	1	1	0	0	0		
R1, R5, R7	1	1	0	0	0	2	
COMBINED COUNT CC						**4**	**PRUNED**

UPPER PART							STATUS
ITEM	**T1**	**T2**	**T3**	**T4**	**T5**	**BC**	
R1	1	1	0	1	1		
R7	0	1	0	1	1	&	
R1, R7	0	1	0	1	1	3	
LOWER PART							
R1	1	1	1	1	0		
R7	1	1	0	0	0	&	
R1, R7	1	1	0	0	0	2	
COMBINED COUNT CC						**5**	**RETAINED**

The itemsets {R1, R5} and {R1, R7} are frequent itemsets. The overall frequent itemsets that are finally discovered is shown in the following table 6.

ITEMSET	COUNT	ITEMSET	COUNT
R1, R3	8	R3, R5	6
R1, R3, R5	6	R3, R7	5
R1, R5	6		
R1, R7	5		

Table 6: Finally discovered frequent itemsets

Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 04, April-2025

RESULTS AND DISCUSSION

The BIRA algorithm was implemented in Java on a Windows 11 personal computer equipped with a 2.66GHz Intel I7 processor, a 1TB DISK, and 16GB of RAM. The evaluation was performed using datasets such as Connect and Accident along with few synthetic datasets generated from IBM quest tool to check the efficiency regarding the memory consumption and execution time consumption. The experiments were conducted using varying minimum support values and the results are shown in the following figure.

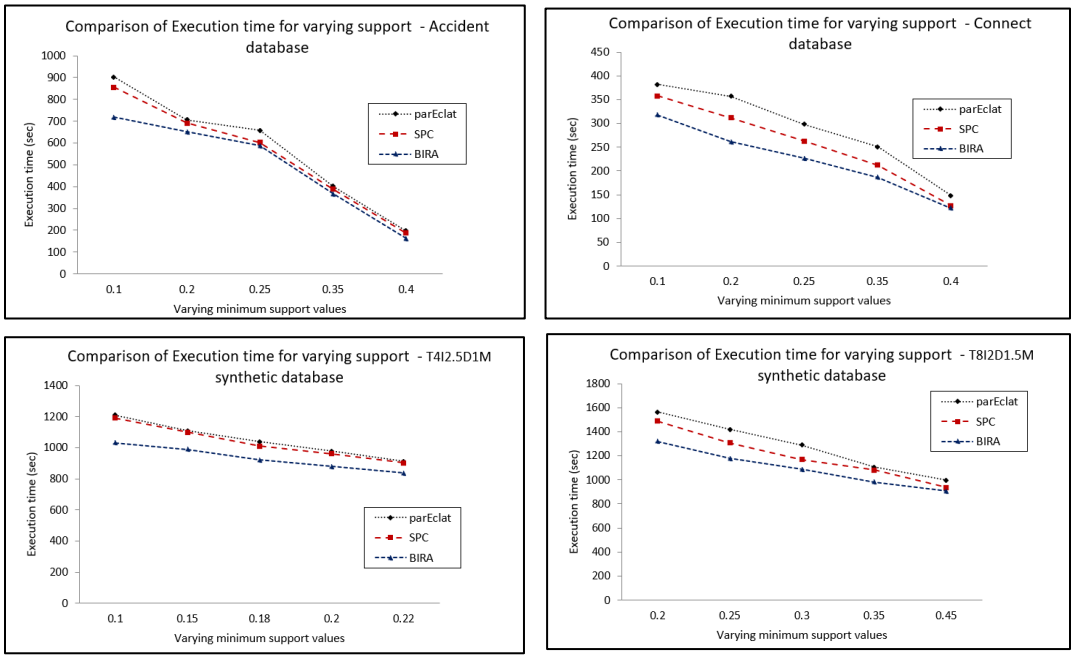

Figure 4: Execution time comparison with varying support values

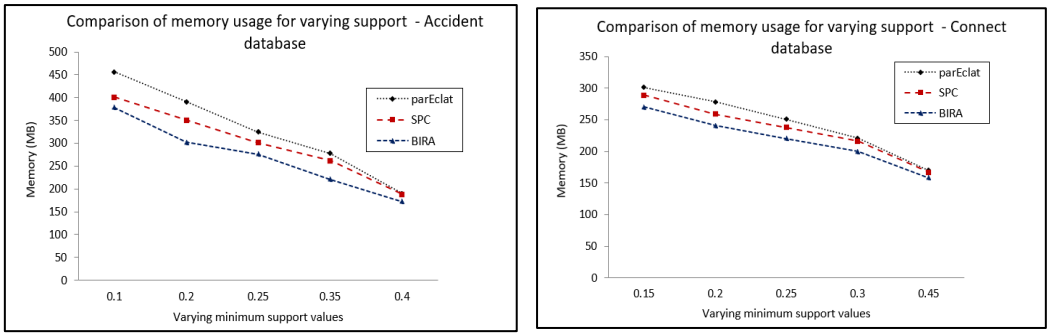

Figure 5: Memory usage comparison with varying support values on real dataset

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 04, April-2025

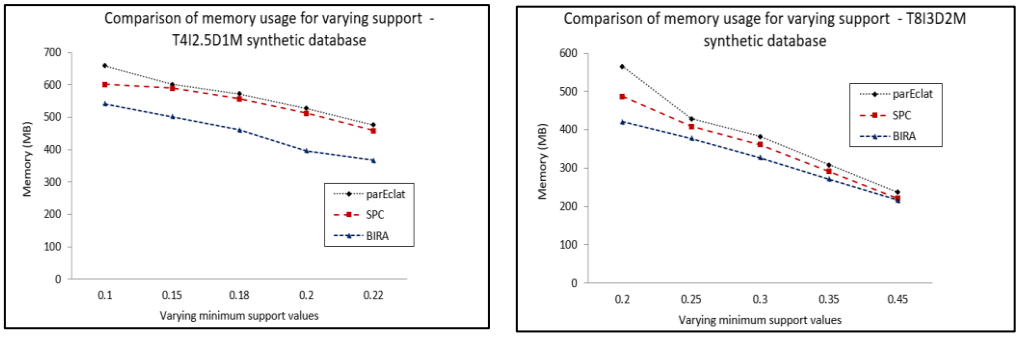

Figure 6: Memory usage comparison with varying support values on synthetic dataset

From the above figures, it is quite clear that the SPC performed reasonably well when compared with the parEclat algorithm and the proposed BIRA outperformed the other two algorithms by a good margin with respect to time and memory consumption.

As the dataset size increased during execution, both the SPC algorithm and the proposed BIRA algorithm continued to perform efficiently, maintaining reasonable processing times and completing their tasks successfully. In contrast, the parEclat algorithm faced considerable challenges. It struggled to handle the larger dataset and ultimately failed to complete the execution. Due to this failure, it was not possible to measure parEclat's memory usage, highlighting its limitations in scalability and memory management when dealing with large datasets. This underscores the advantage of algorithms like SPC and the proposed approach in handling larger data volumes more effectively.

The proposed BIRA algorithm exhibited a significant performance advantage, surpassing the existing algorithms by 30 to 40%. It consistently outperformed all other algorithms in terms of both execution time and processing speed. This notable improvement highlights the algorithm's efficiency in handling computational tasks, making it a more effective solution compared to its counterparts. The enhanced speed and reduced execution time allowed the proposed algorithm to complete tasks more quickly and efficiently, positioning it as a superior option for performance-critical applications.

The tid-list strategy employed by the parEclat algorithm proved to be considerably less effective compared to the proposed algorithm and the SPC algorithm. In terms of performance, parEclat fell notably behind these two algorithms, struggling to keep up with their speed and efficiency. This disparity underscores the limitations of the tid-list approach in parEclat, making it less competitive when compared to the more advanced methods used in the proposed and SPC algorithms. As a result, parEclat was unable to match the overall performance and capabilities demonstrated by the other two algorithms.

CONCLUSION

This paper introduces the BIRA algorithm, which effectively resolves several key shortcomings present in existing algorithms. The experimental results clearly demonstrate that the BIRA algorithm generates a significantly smaller number of candidates due to its advanced and efficient pruning mechanism. By minimizing the number of candidates, the algorithm is able to reduce the overall runtime and memory usage, which in turn leads to a considerable improvement in execution speed. These enhancements make the BIRA algorithm far more efficient and capable of handling larger datasets than traditional algorithms. As a result, BIRA stands out as a more robust and scalable solution, offering superior performance in both computational efficiency and resource management.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 04, April-2025

ABOUT THE AUTHORS

The author Harshitha Pugazhendhi pursuing Final Year Engineering (CSE branch) from Sri Bharathi Engineering College for Women, Kaikuruchi, Pudukkottai District, Tamil Nadu, INDIA is an individual with a strong desire to excel in the ever-evolving field of Data Science, Machine Learning and Artificial Intelligence.

The author Saheena Begam.A pursuing Final Year Engineering (CSE branch) from Sri Bharathi Engineering College for Women, Kaikuruchi, Pudukkottai District, Tamil Nadu, INDIA an aspirant who is driven to succeed in the rapidly changing fields of AI, Deep Learning, and Data Science.

REFERENCES

[1] R. Agarwal and R. Srikant. Mining Sequential Pattern. In Proc. 1995 Int. Conf. Data Engineering, pages 3-10, 1995.
[2] Mohammed J. Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. Parallel algorithms for discovery of association rules. Data Mining and Knowledge Discovery, 1(4):343-373, 1997.
[3] Ming-Yen Lin, Pei-Yu Lee, and Sue-Chen Hsueh. Apriori-based frequent itemset mining algorithms on mapreduce. In Proceedings of the Sixth International Conference on Ubiquitous Information Management and Communication, ICUIMC '12, pages 76:1-76:8, New York, NY, USA, 2012. ACM.
[4] N. Aryabarzan, B. Minaei-Bidgoli, and M. Teshnehlab, "negFIN: An efficient algorithm for fast mining frequent itemsets," Expert Systems with Applications, vol. 105, pp. 129–143, Sep. 2018.
[5] R. Davashi, "UP-tree and UP-mine: A fast method based on upper bound for frequent pattern mining from uncertain data," Engineering Applications of Artificial Intelligence, vol. 106, p. 104477, Nov. 2021.
[6] J. S. P. Poovan, D. A. Udupi, and N. V. S. Reddy, "A multithreaded hybrid framework for mining frequent itemsets," International Journal of Electrical and Computer Engineering (IJECE), vol. 12, no. 3, p. 3249, Jun. 2022.
[7] S. S. Waghere, P. RajaRajeswari, and V. Ganesan, "Retrieval of frequent itemset using improved mining algorithm in hadoop," in Advances in Intelligent Systems and Computing. Springer Singapore, Jul. 2020, pp. 787–798.
[8] C. Zhang, P. Tian, X. Zhang, Z. L. Jiang, L. Yao, and X. Wang, "Fast eclat algorithms based on minwise hashing for large scale transactions," IEEE Internet of Things Journal, vol. 6, no. 2, pp. 3948–3961, Apr. 2019.