A Novel Approach to Enhance the Efficiency of Apriori Algorithm

Erandi Herath
Department of Computer Engineering
University of Sri Jayewardenepura
Colombo, Sri Lanka
erandikghs@gmail.com

Udaya Wijenayake
Department of Computer Engineering
University of Sri Jayewardenepura
Colombo, Sri Lanka
udayaw@sjp.ac.lk

***Abstract*—Data mining is the process of obtaining valuable or significant information from a large-scale database. One significant area of research in the field of data mining is association rules mining. Apriori algorithm is one of the classical algorithms in the association rule mining field. This research analyses the basic ideas and shortcomings of the Apriori algorithm and compares several different styles of its major improvement strategies. Then it suggests an improved version of the Apriori algorithm that utilizes a dataset summarization method, an optimized database mapping technique, an intersection operation, and a joined optimization strategy. These enhancements aim to address the low performance and efficiency by reducing the generation of candidate itemsets and minimizing the execution time. This addresses the issues of generating numerous candidate itemsets and repeatedly scanning the transaction database. After implementing the optimized algorithm, to verify its effectiveness, it has been applied to a groceries dataset, which is for market basket analysis. The improved Apriori algorithm demonstrated significant enhancements over the original algorithm in terms of reduced candidate itemsets and running time, leading to improved algorithm efficiency.**

***Index Terms*—Data mining, Association rules, Apriori algorithm, Market basket analysis**

I. Introduction

Data mining is the study of identifying relevant patterns in large amounts of complicated data while also obtaining crucial information. One significant area of research in the field of data mining is association rules mining. Association rule mining is a procedure which aims to find recurring patterns, correlations, or relationships in datasets. Association rules capture the inherent relationships between data items, revealing their interconnectedness and dependencies [1]. Frequent items are found in association rule mining applications in key technologies and procedures. The goal of frequent itemsets mining is to extract very frequent itemsets from massive volumes of data. Its most common use is to discover new rules in the sales transaction databases that reflect the patterns of customer purchasing behavior, such as the effects on other goods after purchasing a particular kind of goods [2]. This is known as market basket analysis.

Apriori algorithm is one of the classical algorithms in the association rule mining field. It was first proposed by Agrawal and Srikant in 1993 [3]. This algorithm is based on a breadth-first search and has a straightforward, understandable data structure. In particular, Apriori employs an iterative layer-by-layer search strategy, using $(k-1)$-dimensional frequent itemsets (L_{k-1}) to explore k-dimensional frequent itemsets (L_k) [4]. The algorithm has its unique advantages in looking for frequent itemsets. The process of deriving L_k from L_{k-1} consists of two steps: the join step and the prune step.

- The join step: combine the itemsets in L_{k-1} to generate possible candidate itemsets, which is denoted as C_k where $L_k \subseteq C_k$ [4].
- The prune step: scan the database to get the count of each and every candidate itemset. When the count falls below the required level of support, it should be removed from the candidate itemsets [4].

Although the Apriori method has undergone substantial refining, it still has a high space and time complexity and low efficiency. In general, it is challenging to avoid the following defects:

- Produce large number of candidate itemsets: When employing frequent $(k-1)$-itemsets to generate all of the candidate k-itemsets (C_k), many connections are needed and there are a lot of comparisons. If the set of a frequent item set produced by a candidate item set is L_k, the temporal complexity of connection constraints comparison is $\mathcal{O}(kx^2)$ [5]. .
- Scan the transaction database repeatedly: The database should be examined numerous times to gather the support of each proposed k-item set. The database is searched at least n times where $k \,\epsilon\, Z^+; k \leq n$ [5].
- Time consumption when matching candidate itemsets and transaction patterns: For $c \,\epsilon\, C_k$, every $(k-1)$-itemsets in c should be tested whether it is in L_{k-1} or not [4]. Under the best-case scenario, L_{k-1} can meet the requirements with just one scan. But in the worst-case scenario, the requirements will not be met until scanning L_{k-1}, k times [5].

This research analyses the basic ideas and shortcomings of Apriori algorithm and compare several different styles of the major improvement strategies. This research proposes an enhanced Apriori algorithm that addresses the low performance and efficiency of the original algorithm. The improved approach reduces the generation of candidate itemsets and avoid repetitive scanning of the transaction database. To verify the effectiveness of the optimized algorithm, it is applied to a dataset, which is for market basket analysis and the results are compared.

II. Related Work

Since the Apriori algorithm was first developed, many scholars have conducted extensive studies and have presented several practical ways for improvement. The major techniques are hash-based, transaction reduction, database mapping and sampling.

A. Hash-Based Technique

The k-itemsets and associated count are generated by this approach using a hash-based structure known as a hash table. It creates the table using a hash function [6].

Chang and Liu [7] introduced an improved algorithm that directly generates L_2 from a single scan of the database, eliminating the need to generate C_1, L_1, and C_2. Additionally, they replaced the hash tree with a hash table to reduce the cost of searching. They also employed an efficient horizontal data representation and optimized storage strategy, resulting in time and space savings.

Doshi and Joshi [8] also proposed a hashing algorithm to enhance the effectiveness of the Apriori algorithm. This involved utilizing a dictionary (hash table) to store candidate itemsets as keys, while the corresponding value represented the count of their occurrences.

B. Trasasaction reduction

With this technique, fewer transactions must be scanned throughout each cycle. Transactions that do not include frequently used items are flagged or deleted [6].

Cheng et al. [9] introduced a solution to address two issues commonly found in security audit systems: low-level intelligence and underutilization of audit logs. They have proposed the E-Apriori algorithm which uses only the transaction reduction technique. The proposed solution involved the use of association rule mining to create a secure audit system.

Chen et al. [4] proposed BE-Apriori algorithm, which is based on two strategies: the transaction reduction strategy and pruning optimization strategy. In pruned optimization strategy, they have reduced the items in L_{k-1}, which is used to generate L_k. In the transaction reduction strategy, the transactions that need to be scanned have been reduced.

C. Database partitioning

The common itemsets are mined using database partitioning with just two database scans. It states that each itemset must be frequent in at least one database partition in order for it to be considered potentially frequent in the database [6].

Yang et al. [10] introduced the SIAP algorithm which uses the database partitioning and mapping technique. In the first phase of this method, transactions were cut into n blocks horizontally and distributed to m nodes. By running the IAP algorithm n times on m nodes, all the local frequent itemsets are found. In the second phase, global frequent itemsets were generated.

Subithra and Dhenakaran [11] introduced a partitioning algorithm that divides a transactional dataset (D) into n partitions, D_1 to D_n. This algorithm streamlines the dataset processing by performing two main steps. In the first step, the algorithm identifies all itemsets within each partition, collecting the frequent itemsets specific to each partition. In the second step, these global itemsets are counted to determine their significance across the entire dataset. By dividing the dataset into partitions and leveraging these two processes, the partitioning algorithm enhances the efficiency of discovering frequent itemsets.

D. Database mapping

In database mapping technique, the original transaction database will be mapped in a way such that it will optimize the database scanning process [6].

In 2013, Zhang et al. [5] proposed the OApriori algorithm, which is based on the transaction reduction technique and database mapping technique. In this method, they have optimized a mapping method of the database and the connection setup. By using this new data mapping approach, the database is scanned only once. As a result, the frequent itemsets can be obtained simply by scanning the C_k and the generated L_k. This approach effectively addresses the issue of multiple scans, improving the efficiency of the Apriori algorithm.

Du et al. [12] proposed DC_Apriori algorithm, which used the transaction reduction technique, database partitioning method and database mapping method [12]. It restructures the storage structure of the database and improved the connection of frequent itemsets. The generation of L_k was only needed to join the L_1 with L_{k-1}. The running time of DC_Apriori was significantly less than the Apriori algorithm.

E. Sampling

This technique involves randomly selecting a sample (S) from the database (D) and searching for frequent itemsets within S. However, when using sampling, there is a possibility of losing common itemsets that are globally used. To mitigate this, the minimum support value can be decreased [6].

Krishna and Amarawat [13] proposed a sampling-based approach to improve the performance of the Apriori algorithm. The algorithm utilizes random sampling from the database to identify frequent itemsets. It works by selecting a random sample (S) and finding frequent itemsets within that sample using a lower support threshold. This approach provides an efficient way to handle large databases that do not fit into main memory by working with smaller partitions or samples.

Umarani and Punithavalli [14] introduced a novel and effective progressive sampling-based approach specifically designed for mining association rules in large databases. Initially, they used the Apriori algorithm to extract frequent patterns from an initial sample, which is carefully chosen based on temporal characteristics and database size. Using the generated frequent itemsets, they obtained and sorted the negative border of the initial sample. Next, they scanned the midpoint itemset in the sorted negative border against the concrete database to determine its frequency. Their experimental results demonstrated the efficiency and effectiveness of the proposed progressive sampling approach in association rule mining.

F. Other approaches

In the year 2020, Karimtabar and Fard suggested Intelligent Apriori (IAP) algorithm which uses a matrix-based method [15]. This technique prevents the creation of unnecessary itemsets when constructing a new matrix, which reduces the number of transaction scans required to produce frequent itemsets [15]. The results of their experiment showed that the proposed algorithm significantly reduces the number of transaction scans required to find frequent itemsets.

Yang [16] proposed AC-Apriori algorithm which uses a new technique "AC automation" [16]. They have introduced the AhoCorasick (AC) automation method which converts the process of traversing a transaction to a multimode match. The AC-Apriori algorithm was a combination of two techniques: the Apriori algorithm and the Aho-Corasick automation. It aimed to reduce the number of times a transaction database needs to be scanned while still achieving the same mining results.

Despite significant refinements proposed by previous researchers, the Apriori algorithm still continues to exhibit high space and time complexity, leading to low efficiency. Therefore, we introduce a novel strategy, which combines a utilization of a summarized dataset, an optimized database mapping technique, an intersection operation, and a new joined optimization strategy. Prior to our research, no one has extensively explored a combined method similar to ours. Through our research, we demonstrate the effectiveness of the proposed method in enhancing the Apriori algorithm and propose it as a valuable alternative in association rule mining tasks.

III. Proposed Apriori Algorithm

In this research we propose several improvements for Apriori algorithm to increase its efficiency.

A. Summarized dataset

In this research, we have introduced a new data summarization method. For the experiment, we used a groceries dataset which is for market basket analysis. It consisted of three columns which are member, date and item description. This dataset consisted of 38765 rows, including purchase orders of people from grocery stores. The data set is available in Kaggle [17].

After summarizing the original dataset, the summarized dataset focused on essential details, including three columns: items, transactions, and available sizes. The item column contained item details, the transaction column held relevant transaction ids (TIDs), and the available sizes column indicated the quantity of items per transaction.

Generating the summarized dataset is a one-time process, more time-efficient than the original Apriori algorithm. In the original algorithm, generating k-dimensional frequent itemsets involves repeated scans of the entire database, leading to significant time consumption. The optimized algorithm addresses this inefficiency by extracting only necessary information from the original dataset, avoiding repetitive scans and reducing computational time.

B. Optimized database mapping technique

After summarizing the original database, we introduced a new database mapping technique. We used four linear tables for this purpose. The first table stored the frequent itemsets, the second table stored the relevant transaction IDs (TID) of the frequent itemsets, the third table stored the infrequent itemsets, and the fourth table stored the available sizes of the itemsets.

C. Introduction of intersection operation

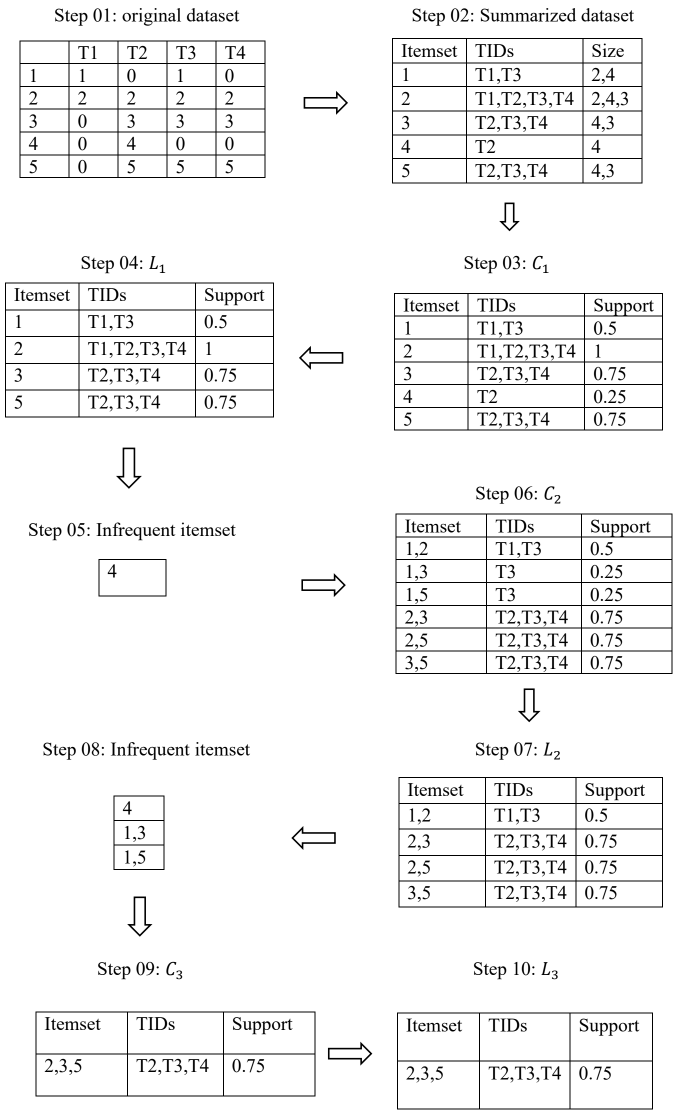
When generating candidate k-item sets from frequent $(k-1)$-item sets, we performed the intersection operation on the corresponding transaction number sets. This operation allowed us to generate new transaction number sets by obtaining the intersection of the transaction sets from the itemsets that were being joined.

D. Joined optimization strategy

In our research, we introduced a size-checking method for itemsets to optimize the generation of candidate itemsets. This method involved a careful examination of each item in the two itemsets being combined, ensuring that their sizes were greater than or equal to the specified threshold value denoted as k. If any item failed to meet this size requirement, all itemsets containing that specific item were excluded from consideration in C_k. This approach effectively filtered out itemsets that did not meet the size criteria, focusing on relevant and meaningful itemsets for further analysis.

During the generation of candidate itemsets, we implemented a check for at least one common transaction between the itemsets being joined. If no common transactions were found, it signaled that generating candidate itemsets using those specific itemsets would be futile. Consequently, candidate k-itemsets were not generated from those itemsets, contributing to a more efficient and streamlined process.

The introduction of the infrequent itemset: During the pruning step, we kept another linear table to store the infrequent itemsets. These infrequent itemsets were characterized by having a support value less than the minimum support count. When generating candidate itemsets, we conducted checks on these infrequent itemsets. Specifically, we examined if any subsets of the two itemsets being joined were present in the infrequent itemset collection. If we found such subsets, using them candidate itemsets were not generated. Consequently, all candidate itemsets containing these subsets were disregarded and ignored in the subsequent analysis.



Step 01: original dataset

	T1	T2	T3	T4
1	1	0	1	0
2	2	2	2	2
3	0	3	3	3
4	0	4	0	0
5	0	5	5	5

Step 02: Summarized dataset

Itemset	TIDs	Size
1	T1,T3	2,4
2	T1,T2,T3,T4	2,4,3
3	T2,T3,T4	4,3
4	T2	4
5	T2,T3,T4	4,3

Step 03: C_1

Itemset	TIDs	Support
1	T1,T3	0.5
2	T1,T2,T3,T4	1
3	T2,T3,T4	0.75
4	T2	0.25
5	T2,T3,T4	0.75

Step 04: L_1

Itemset	TIDs	Support
1	T1,T3	0.5
2	T1,T2,T3,T4	1
3	T2,T3,T4	0.75
5	T2,T3,T4	0.75

Step 05: Infrequent itemset

4

Step 06: C_2

Itemset	TIDs	Support
1,2	T1,T3	0.5
1,3	T3	0.25
1,5	T3	0.25
2,3	T2,T3,T4	0.75
2,5	T2,T3,T4	0.75
3,5	T2,T3,T4	0.75

Step 07: L_2

Itemset	TIDs	Support
1,2	T1,T3	0.5
2,3	T2,T3,T4	0.75
2,5	T2,T3,T4	0.75
3,5	T2,T3,T4	0.75

Step 08: Infrequent itemset

4
1,3
1,5

Step 09: C_3

Itemset	TIDs	Support
2,3,5	T2,T3,T4	0.75

Step 10: L_3

Itemset	TIDs	Support
2,3,5	T2,T3,T4	0.75

Fig. 1. Table-based execution process of the improved Apriori algorithm, showcasing enhancements for candidate itemsets and generating frequent itemsets

E. Execution process of improved algorithm

In Fig. 1, the initial dataset is introduced in Step 01, serving as the starting point for subsequent analytical steps. Following a summarization process in Step 02, a refined dataset is obtained. The construction of a distinct subset, C_1, is illustrated in Step 03, where the support count for each itemset is calculated. Items with a support count below the minimum threshold (set at 0.3) are systematically eliminated, leading to the formation of L_1 in Step 04. Simultaneously, the infrequent itemset is updated with items below the minimum support (Step 05).

The subsequent phase involves generating k-dimensional itemsets from $k-1$ dimensional ones. For instance, in Step 06, 2-dimensional candidate itemsets, C_2, are derived using the 1-dimensional frequent itemset. During this process, itemsets with a size less than two are excluded, and conditions regarding common transactions and infrequent itemsets are checked (here $k-1=1$, so $k=2$). This iterative procedure continues until no further frequent itemsets can be ascertained, as demonstrated by the example, concluding with the derivation of L_3 in Step 10.

F. Optimized algorithm

Our optimized algorithm takes the minimum support value ($minsup$) as input and provides the output of frequent itemsets (L_k). This optimized algorithm consists of three essential functions that play a vital role in creating candidate itemsets and detecting frequent itemsets.

1) One-dimensional candidate itemset generation function: This function is responsible for generating the initial candidate itemsets denoted as C_1. Its primary purpose is to create the set of itemsets for the first level of analysis (see Algorithm 1).
2) Frequent itemset generation function: This function plays a key role in generating frequent itemsets. It identifies itemsets that meet a specified minimum support threshold and are deemed significant for further analysis (see Algorithm 2).
3) Candidate itemset generation function: The candidate itemset generation function focuses on generating candidate itemsets for subsequent levels of analysis. It combines frequent itemsets from the previous level to form new potential itemsets for further evaluation (see Algorithm 3).

Algorithm 1 Function to generate one dimensional candidate itemsets

```
1:  procedure CREATEC1()
2:      n=no of rows in summarized dataset
3:      i=0
4:      $C_1$=one dimensional candidate Item set
5:      $S_1$=available Sizes Set
6:      while $i \le n$ do
7:          Add item to $C_1$
8:          Add item to $S_1$
9:      Sort $C_1$
10:     return $C_1$, $S_1$
```

Algorithm 2 Function to generate frequent item sets

```
1:  procedure GENERATEFREQUENTITEMSETS($C_k$, $T_k$,MIN SUP,INFREQUSET)
2:      while not end of the itemsets in Ck do
3:          supCount=size(($T_k$[itemset]))/totalTrans
4:          if supCount>= minsup then
5:              Add itemset to $L_k$
6:              FITrans[itemset]=$T_k$[itemset]
7:              supVal[itemset]=supCount
8:          else
9:              Add itemset to inFrequSet
10:     return $L_k$, FITrans, supVal, inFreqSet
```

Algorithm 3 Function to generate candidate itemsets

```
1: procedure GenerateCandidateSets($L_k, T_k, S_k, k$, inFrequSet)
2:     for $i$ in range(size($L_k$)) do
3:         for $j$ in range($i + 1$, $len(L_k)$) do
4:             $L_1 = L_k[i][: k - 2]$
5:             $L_2 = L_k[j][: k - 2]$
6:             Sort $L_1$
7:             Sort $L_2$
8:             if $L_1 == L_2$ then
9:                 newItemSet = $L_k[i] \cup L_k[j]$
10:                for each item in newItemSet do
11:                    for element in $S_k[item]$ do
12:                        if $k \leq$ element then
13:                            boolCheck.append(True)
14:                            Break the Loop
15:                        else
16:                            Continue
17:                if size of list boolCheck == $k$ then
18:                    tempValueHolder = Lk[i] ∩ Lk[j]
19:                    if tempValueHolder is empty then
20:                        for each itemSet in inFrequSet do
21:                            for item in itemSet do
22:                                for element in newItemSet do
23:                                    if element == item then
24:                                        increment KeepTrack
25:                            if itemSetSize == KeepTrack then
26:                                Check = True
27:                                Break the loop
28:                            else
29:                                itemCount = total item count
30:                        if size of (newItemSet != itemCount) and (check==False) then
31:                            add newItemSet to $C_k$
32:                            newTrans[newItemSet] = tempValHolder
33:     size = Size of $C_k$
34:     return size, $C_k$, newTrans
```

IV. Experiments and Results

In the first experiment, we compared the performance of two versions of the Apriori algorithm across different support values, as illustrated in Fig. 2. The optimized Apriori algorithm exhibited remarkable improvements, reducing execution time to less than half of the original algorithm. The trend also revealed enhanced efficiency as the support count increased, indicating its improved capability to handle larger datasets.

Table I provides a quantitative performance comparison, highlighting execution times and efficiency improvements across various support values. The average efficiency enhancement of 71.65% reflects the typical reduction in execution time achieved with the optimized Apriori algorithm, offering a consolidated measure of its time-saving capabilities in different scenarios.

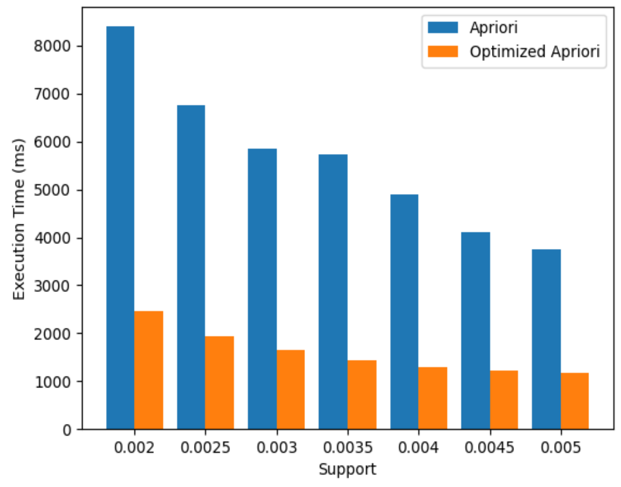


Fig. 2. Comparison of the execution time between original Apriori algorithm and optimized Apriori algorithm under different support values.

The substantial reduction in execution time achieved by the optimized Apriori algorithm aligns with the primary goal of this research, which is to enhance the efficiency of the algorithm. This improvement is crucial for handling large-scale datasets and time-sensitive applications where quick and efficient mining is essential.

The optimized Apriori algorithm offers superior time efficiency, enhancing productivity and decision-making across domains like market basket analysis, recommendation systems, and customer behavior analysis. Rapid data processing enables quicker insights and the identification of valuable patterns and relationships.

In Fig. 3, a visual comparison depicts the number of generated candidate itemsets using the original Apriori algorithm and the optimized version at different support values. The optimized algorithm consistently generates about half the candidate itemsets, showcasing substantial improvement through efficient pruning. This reduction minimizes computational overhead, enhancing speed and efficiency. The streamlined search for relevant itemsets simplifies data mining, facilitating quicker extraction of valuable patterns from large datasets, aligning with the goal of improving scalability and performance.

Apart from the above comparisons, we conducted a validation using the groceries dataset which is for market basket analysis. The comparison of results from both algorithms revealed that they produced the same frequent itemsets. This serves as a strong evidence or confirmation that the optimized algorithm is indeed accurate and reliable in its calculations. Moreover, it demonstrates that efficiency gains were achieved without sacrificing the accuracy.

TABLE I
PERFORMANCE COMPARISON OF APRIORI ALGORITHMS

Support Value	**Time Elapsed (ms)**		**Efficiency Improvement**
	Original Apriori	**Optimized Apriori**	
0.0020	8399.23	2455.87	70.76%
0.0025	6763.69	1942.03	71.29%
0.0030	5854.77	1658.26	71.68%
0.0035	5726.23	1425.64	75.10%
0.0040	4892.51	1298.74	73.45%
0.0045	4108.63	1215.47	70.42%
0.0050	3743.67	1166.56	68.84%
	Average		71.65%

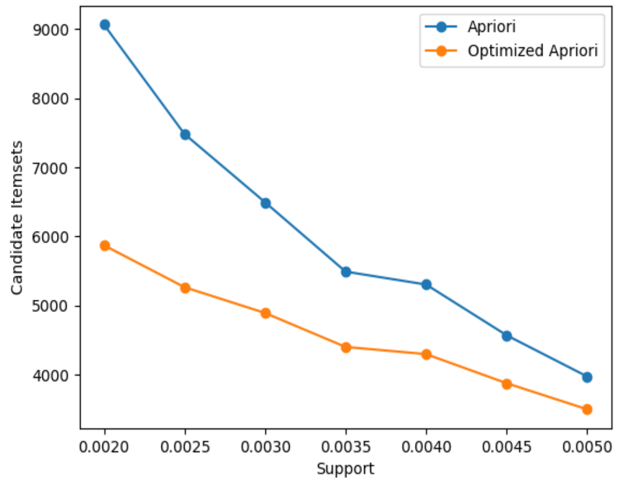


Fig. 3. Comparison between the number of generated candidate itemsets of original Apriori algorithm and optimized Apriori algorithm under different support values.

V. CONCLUSION

This study comprehensively analyzes the fundamental concepts and limitations of the Apriori algorithm, along with exploring notable enhancement approaches. We proposed multiple improvements to reduce execution time and candidate itemsets, enhancing efficiency around 70%. Future enhancements could target memory reduction and feature selection. Optimizing the algorithm holds promise across domains like education, healthcare, E-Commerce, and ecological research. It could recommend courses, detect medication patterns, analyze product associations, and aid ecological studies. These advancements amplify the Apriori algorithm's applicability and effectiveness, contributing to refined association rule mining solutions.

VI. ACKNOWLEDGEMENT

This research was supported by the Science and Technology Human Resource Development Project, Ministry of Education, Sri Lanka, funded by the Asian Development Bank (Grant No. STHRD/CRG/R1/SJ/06).

REFERENCES

[1] W. Nengsih, "A comparative study on market basket analysis and apriori association technique," in *2015 3rd International Conference on Information and Communication Technology (ICoICT)*, 2015.
[2] M. Hossain, A. H. Sattar, and M. K. Paul, "Market basket analysis using apriori and fp growth algorithm," in *2019 22nd International Conference on Computer and Information Technology (ICCIT)*, 2019.
[3] R. Agrawal, T. Imielinski, and A. Swami, "Database mining: A performance perspective," *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 6, pp. 914–925, 1993.
[4] Z. Chen, S. Cai, Q. Song, and C. Zhu, "An improved apriori algorithm based on pruning optimization and transaction reduction," in *2011 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC)*, 2011.
[5] K. Zhang, J. Liu, Y. Chai, J. Zhou, and Y. Li, "A method to optimize apriori algorithm for frequent items mining," in *2014 Seventh International Symposium on Computational Intelligence and Design*, 2014.
[6] Engati, "Apriori algorithm," Engati, accessed: Jun. 17, 2023. [Online]. Available: https://www.engati.com/glossary/apriori-algorithm
[7] R. Chang and Z. Liu, "An improved apriori algorithm," in *Proceedings of 2011 International Conference on Electronics and Optoelectronics*, 2011.
[8] A. J. Doshi and B. Joshi, "Comparative analysis of apriori and apriori with hashing algorithm," *International Research Journal of Engineering and Technology (IRJET)*, vol. 05, Jan 2018.
[9] M. Cheng, K. Xu, and X. Gong, "Research on audit log association rule mining based on improved apriori algorithm," in *2016 IEEE International Conference on Big Data Analysis (ICBDA)*, 2016.
[10] S. Yang, G. Xu, Z. Wang, and F. Zhou, "The parallel improved apriori algorithm research based on spark," in *2015 Ninth International Conference on Frontier of Computer Science and Technology*, 2015.
[11] M. Subithra and S. S. Dhenakaran, "Performance analysis of apriori and partitioning method in frequent itemset generation," *IJRDO - Journal of Computer Science and Engineering*, vol. 2, no. 8, August 2016.
[12] J. Du, X. Zhang, H. Zhang, and L. Chen, "Research and improvement of apriori algorithm," in *2016 Sixth International Conference on Information Science and Technology (ICIST)*, 2016.
[13] B. Krishna and G. Amarawat, "Data mining in frequent pattern matching using improved apriori algorithm," in *Advances in Intelligent Systems and Computing*, 2018, pp. 699–709.
[14] V. Umarani and M. Punithavalli, "Developing novel and effective approach for association rule mining using progressive sampling," in *2009 Second International Conference on Computer and Electrical Engineering*, 2009.
[15] N. Karimtabar and M. J. Shayegan Fard, "An extension of the apriori algorithm for finding frequent items," in *2020 6th International Conference on Web Research (ICWR)*, 2020.
[16] J. Yang, H. Huang, and X. Jin, "Mining web access sequence with improved apriori algorithm," in *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, 2017.
[17] H. Dedhia, "Groceries dataset," https://www.kaggle.com/datasets/heeraldedhia/groceries-dataset, Sep 2020, accessed on Dec. 25, 2022.