ReverseApriori

Yulin Zhou*, Jiaqi Zhang† *Department of Computer Science, Sun Yat-sen University, Guangzhou, China

***Abstract*—Frequent itemset mining is pivotal in data mining with applications such as market basket analysis. The foundational Apriori algorithm, suffers from performance limitations due to its combinatorial nature. We introduces ReverseApriori that reduces the search space by incorporating an intersection approach, thereby pruning non-promising trans-set at each stage, and significantly improves efficiency and scalability. Extensive experiments on real-world datasets demonstrate that our improved algorithm outperforms traditional methods in terms of time and memory.**

***Index Terms*—Frequent itemset mining, Apriori, ReverseApriori, Big Data, Intersection, Itemset-Trans Mapping.**

I. Introduction

IN the burgeoning field of data mining, the extraction of frequent itemsets plays a crucial role in uncovering relationships hidden within vast datasets. Applications of this research span various domains such as market basket analysis, bioinformatics, and intrusion detection, where understanding item co-occurrences can lead to significant insights and economic benefits. The Apriori algorithm, introduced by Agrawal and Srikant in 1994, laid the foundational work for this area by providing a methodical approach to mine these frequent itemsets through candidate generation and support counting.

Despite its widespread adoption, the standard Apriori algorithm faces critical efficiency challenges, particularly with large or dense datasets. The algorithm's efficiency is hampered by its exhaustive itemset generation process, which often results in substantial computational overhead due to the generation of an excessively large number of candidate sets, many of which are not frequent. Additionally, the repetitive scanning of the database to calculate the support of itemsets adds to the computational burden, especially as the size and complexity of data increase.

Over the years, several variations of the Apriori algorithm have been proposed to address these inefficiencies. These variations typically focus on optimizing the candidate generation step, improving the support counting mechanism, or reducing the dataset dynamically during the mining process. However, there remains a need for an approach that can handle contemporary big data applications, which not only require high processing speeds but also demand minimal memory consumption.

This paper introduces an enhanced version of the Apriori algorithm that specifically targets these challenges. Our contributions are twofold: firstly, we propose an optimized candidate generation process that employs a novel difference-set-based technique to minimize the search space effectively, and secondly, we incorporate a dynamic transaction reduction strategy that progressively eliminates transactions from consideration as soon as it is determined that they no longer contain relevant information for future calculations. These innovations collectively improve the performance of the Apriori algorithm, making it more suitable for modern applications involving large-scale data environments.

The remainder of this paper is organized as follows: Section 2 reviews related work in the area of frequent itemset mining and positions our contributions within the existing literature. Section 3 describes the methodology of our enhanced Apriori algorithm in detail. Section 4 presents a comprehensive evaluation of our algorithm against traditional methods using both synthetic and real-world datasets. Finally, Section 5 concludes the paper with a summary of our findings and discussions on potential future research directions.

II. Background

The concept of association rule mining was introduced by R. Agrawal in 1993 [1]. The following year, R. Agrawal and R. Srikant proposed the Apriori algorithm in their published paper [2]. This algorithm is one of the most classic algorithms in the field of association rule mining and is a priori algorithm.

The Apriori algorithm is based on an anti-monotone Apriori heuristic: if any length k pattern is not frequent in the database, its length (k+1) super-pattern can never be frequent. The essential idea is to iteratively generate the set of candidate patterns, generating k-itemsets from (k-1)-itemsets. The First pass of the algorithm simply counts item occurrences to determine the large 1-itemsets. A subsequent pass, say pass k, consists of two prases. First, the large itemsets L_{k-1} found in the (k-1)th pass are used to generate the candidate itemsets C_k. Next, the database is scanned and the support of candidates in C_k is counted.

The Apriori algorithm achieves good performance gain by reducing the size of candidate sets. However, in situations with large datasets and quite low minimum support thresholds, the performance of the Apriori algorithm can be very poor due to the following reasons:

- It is tedious to repeatedly scan the database and check a large set of candidates by pattern matching, which is especially true for mining long patterns.
- The number of candidate sets generated by the Apriori algorithm can be very large, and many of these candidate sets may ultimately not be frequent. It is costly to handle a huge number of candidate sets.

Due to the limitations of the Apriori algorithm at the time, the PCY algorithm was proposed by Park, Chen, and Yu [3]. Its goal is to reduce the frequent database scans of the Apriori algorithm and decrease the memory occupied by candidate itemsets. On one hand, they applied hash functions to frequent item mining; on the other hand, during the first scan of the transaction database, they used the remaining space to store the hash table, thus reducing the large amount of space required during the second scan. However, in many cases, the optimization effect of the PCY algorithm on the Apriori algorithm is negligible. Moreover, when searching for frequent 3-itemsets and higher-order itemsets, the PCY algorithm does not provide any optimization.

Meanwhile, scholars have built on the above work and proposed algorithms such as the multi-stage [4] algorithm and multi-hash [5] algorithm. Compared to PCY, these algorithms made improvements on different levels, but they still face the challenge of needing to frequently scan the database.

In 2000, to overcome the issues mentioned above, Han and Pei et al. [6] proposed the well-known FP-growth algorithm in 2000. Compared to the traditional Apriori algorithm, the FP-growth algorithm was a significant technological revolution in the history of association rule mining, greatly surpassing the traditional Apriori algorithm in terms of efficiency. It is based on a tree data structure that represents all frequent itemsets with a tree, called an FP-tree. It only requires two scans of the database, avoiding the generation of a large number of candidate sets. By trading space for time, the algorithm is highly efficient. However, if a tree has too many child nodes, such as when a tree containing only prefixes is generated, the algorithm's efficiency will significantly decrease. The FP-Growth algorithm requires recursively generating conditional databases and conditional FP-trees, resulting in high memory overhead, and it can only be used for mining single-dimensional Boolean association rules.

III. Method

The motivation for proposing this improved algorithm is that the Apriori algorithm requires repeated scans of the database to calculate the support of itemsets, which consumes a significant amount of time. We aim to propose a new method to reduce this overhead when mining frequent itemsets.

Instead of recording the support of an itemset over the entire transaction set, we can directly record the distribution of the itemset over the transaction set, i.e., the set of transactions in which the itemset appears. This way, when determining whether an itemset is frequent, we only need to perform an intersection operation on the distributions corresponding to the two parts of the itemset used to synthesize it, to obtain the distribution of the itemset. If the length of this distribution is greater than or equal to the minimum support threshold, then the itemset is frequent.

Specifically, as shown in Algorithm 1, the first pass of the algorithm simply records the distribution of items over the transaction set and obtains L_1 based on the length of the distribution. A subsequent pass, say pass k, Generate $dist_{k+1}$ using $dist_k$ and $dist_1$. During the generation process, we iterate over the pairs of L_k and L_1. If the last item in the L_k item is lexicographically smaller than the item in the L_1 item, and the length of the intersection of their distributions is greater than or equal to the minimum support threshold, then the combined $dist_{candidate}$ is added to the $dist_{k+1}$ set.

In this way, we only need to scan the database once, iteratively generating frequent $dist_{k+1}$ from $dist_k$ and $dist_1$ layer by layer as shown in Algorithm 2.

Algorithm 1 ReverseApriori

```
Input :
    T: transaction set
    threshold: Minimum support threshold
Output :
    Frequent itemsets written to files
 1: $k$ = 1
 2: $dist$ = {distribution of frequent 1-itemsets}
 3: $dist_k = dist$
 4: while $dist_k \neq \emptyset$ do
 5:     $dist_{k+1} = \text{Construct}(dist, dist_k, threshold)$
 6:     if $dist_{k+1} \neq \emptyset$ then
 7:         write $dist_{k+1}$.keys to file
 8:     end if
 9:     $dist_k = dist_{k+1}$
10:     $k = k + 1$
11: end while
```

Algorithm 2 Construct $dist_{k+1}$

```
Input :
    dist: Distribution of frequent 1-itemsets
    $dist_k$: Distribution of frequent k-itemsets
    threshold: Minimum support threshold
Output :
    $dist_{k+1}$: Distribution of frequent (k+1)-itemsets
 1: $L_1 =$ keys of $dist$
 2: $L_k =$ keys of $dist_k$
 3: $dist_{k+1} = \{\}$
 4: for each $itemset \in L_k$ do
 5:     for each $item \in L_1$ do
 6:         if $itemset[-1] < item$ then
 7:             $dist_{candidate} = dist_k[itemset] \cap dist[item]$
 8:             if $|d| \geq threshold$ then
 9:                 $dist_{k+1}[itemset \cup item] = dist_{candidate}$
10:             end if
11:         end if
12:     end for
13: end for
14: return $dist_{k+1}$
```

Example. Consider the database in Table I and assume that minimum support is 3 transactions.

TABLE I: Database

TID	Items
1	a,b,c,e
2	a,c,e,f
3	b,c,d,e
4	a,b,d,f
5	a,c,d,e

TABLE II: $dist_1$

Itemset	Distribution
{a}	{1,2,4,5}
{b}	{1,3,4}
{c}	{1,2,3,5}
{d}	{3,4,5}
{e}	{1,2,3,5}
{f}	{2,4}

IV. Experiment

When testing the algorithm, we used shopping datasets with 9,835 and 98,350 purchase records respectively. Since the computational results of the algorithm are identical to those of

TABLE III: $dist_2$

Itemset	Distribution
{a,c}	{1,2,5}
{a,e}	{1,2,5}
{c,e}	{1,2,3,5}

TABLE IV: $candidate_3$

Itemset	Distribution
{a,c,d}	{5}
{a,c,e}	{1,2,5}
{a,c,f}	{2}
{a,e,f}	{2}
{c,e,f}	{2}

the Apriori algorithm, we omitted the correctness evaluation here. The focus of the tests was mainly on the inference efficiency of the algorithm.

A. Experimental Environment

- **Model**: Lenovo Legion R7000P 2021
- **Operating System**: Windows 11
- **Processor**: AMD Ryzen 7 5800H
  - 8 cores / 16 threads
  - Base frequency: 3.2 GHz
  - Max boost frequency: 4.4 GHz
- **Graphics Card**: NVIDIA GeForce RTX 3060
  - Video memory: 6GB GDDR6
- **Memory**: 16GB DDR4 3200MHz
  - Dual channel
- **IDE**:
  - PyCharm
- **Libraries and Frameworks**:
  - **Python**:
    - itertools
    - PyTorch
    - NumPy
    - Pandas
    - tracemalloc
    - time

B. Testing Datasets

The datasets used in this experiment consist of two purchase record datasets. One is obtained from real-world data, and the other is randomly generated. However, as mentioned earlier, since this experiment primarily focuses on the time efficiency of the algorithm, the use of a randomly generated dataset does not affect the experiment itself.

C. Experimental Results

The following two tables list the execution times of our algorithm, the corresponding baseline algorithms, and the current state-of-the-art algorithms on smaller and larger datasets respectively, for performance comparison. It can be seen that our algorithm achieves similar time efficiency to the current mainstream FP-tree algorithm on the smaller dataset, and significantly outperforms the FP-tree algorithm on the larger dataset. Additionally, it exhibits a dominant advantage over the basic Apriori algorithm in terms of execution time.

In addition, The figures below(Fig.1 and Fig.2) shows the time we should consume per iteration when used for two kinds of datasets.

TABLE V: Execution Time Efficiency on Small Dataset(9835)

Algorithm	**Threshold=3**	**Threshold=4**	**Threshold=5**
Apriori	3624 s	2897 s	2355 s
FPtree	3.64 s	2.66 s	2.49 s
ReverseApriori	3.79 s	2.55 s	1.66 s

TABLE VI: Execution Time Efficiency on Big Dataset(98350)

Algorithm	**Threshold=3**	**Threshold=4**	**Threshold=5**
Apriori	>1day	>1day	>1day
FPtree	26.82 s	24.79 s	27.62 s
ReverseApriori	18.82 s	14.10 s	13.33 s

V. Conclusion

Our proposed frequent itemset mining algorithm, ReverseApriori, aligns with the original Apriori algorithm in terms of correctness and applicable scenarios. However, the algorithm presented in this paper offers a significant advantage in execution time over the traditional Apriori algorithm. Moreover, even when compared with state-of-the-art algorithms like FP-tree, our proposed algorithm demonstrates clearly superior performance. In an era characterized by large data volumes, many datasets that require processing and analysis are undoubtedly better suited to our algorithm, which provides faster inference. Additionally, in scenarios involving small datasets, our inference can achieve real-time analysis, making our algorithm applicable in a broader and more profound range of situations compared to many earlier algorithms.

References

[1] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pp. 207–216, 1993.

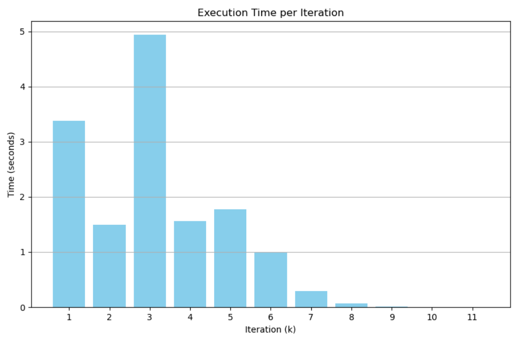
[2] R. Agrawal, R. Srikant, *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, pp. 487–499, Santiago, 1994.

[3] J. S. Park, M.-S. Chen, and P. Yu, "Using a hash-based method with transaction trimming for mining association rules," *TKDE*, vol. 9, no. 5, pp. 813–825, 1997.

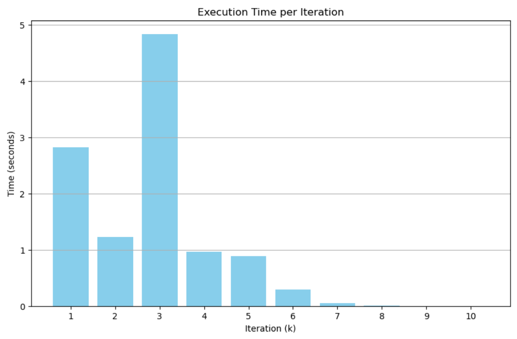
[4] R. J. and J. Bayardo, "Efficient mining of long patterns from databases," *SIGMOD*, pp. 85–93, 1998.

[5] J. S. Park, M.-S. Chen, and P. S. Yu, "An effective hash-based algorithm for mining association rules," *SIGMOD*, pp. 175–186, 1995.

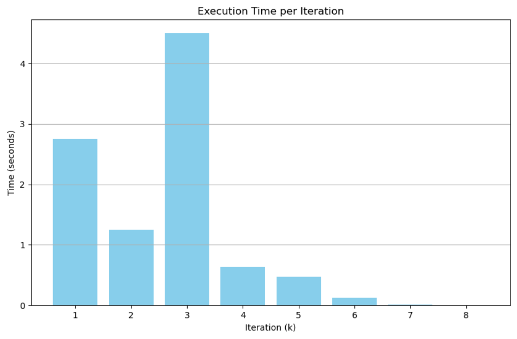
[6] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM sigmod record*, vol. 29, no. 2, pp. 1–12, 2000.



(a) Time Consumed per Iteration when Threshold=3.

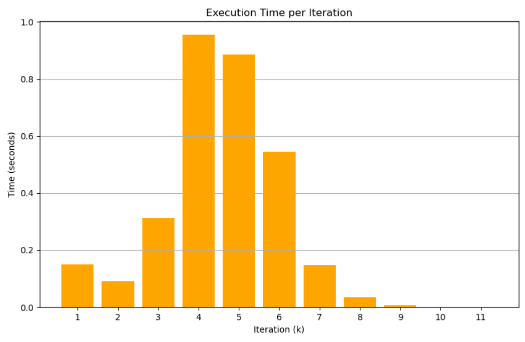


(b) Time Consumed per Iteration when Threshold=4.

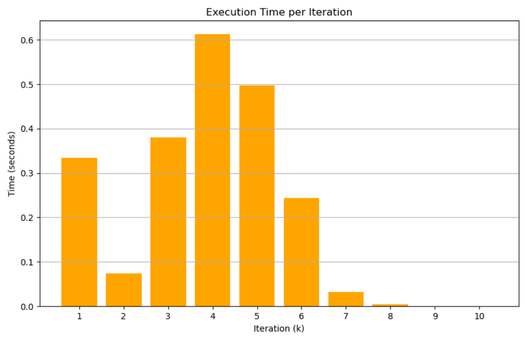


(c) Time Consumed per Iteration when Threshold=5.

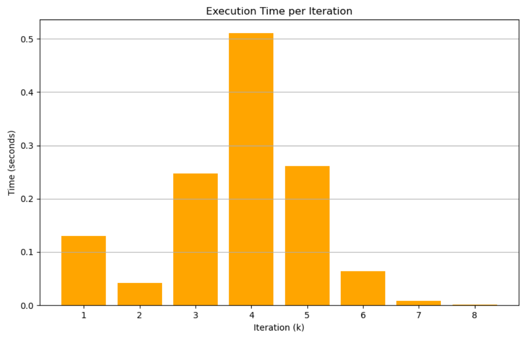
Fig. 1: Time Consumed per Iteration when Used for Big Dataset.



(a) Time Consumed per Iteration when Threshold=3.



(b) Time Consumed per Iteration when Threshold=4.



(c) Time Consumed per Iteration when Threshold=5.

Fig. 2: Time Consumed per Iteration when Used for Small Dataset.