Enhancing FP-Growth: Efficient Projection Storage in Large-Scale Data Mining

1023040919 Huangwenqing Shi
School of Computer Science
Nanjing University of Posts and Telecommunications
Nanjing, P.R.C

***Abstract*—In the field of data mining, efficient processing of large volumes of data is crucial, and frequent itemset mining plays a key role. The FP-Growth algorithm is a critical algorithm in this domain, but due to its storage-intensive nature, it faces significant challenges. This paper analyzes the limitations of the FP-Growth algorithm, introduces an improved algorithm, ImFP-Growth (Improved FP-Growth Algorithm), which adjusts the processing order and projection direction to enhance the FP-Growth algorithm, and provides a detailed exposition of the algorithm's principles and framework. Both the FP-Growth and ImFP-Growth algorithms are employed for data mining on the target dataset. Extensive experiments are conducted to analyze the performance of the FP-Growth and ImFP-Growth algorithms, comparing their performance across different minimum support intervals. The experiments demonstrate that the algorithms mentioned in this paper can achieve correct results, and the ImFP-Growth algorithm achieves a significant improvement in elapsed time.**

***Index Terms*—data mining, FP-Growth, FP-Trees, frequent itemset mining**

I. Introduction

In recent decades, with the advancement of computer technology, information technology has rapidly evolved, and computers have been extensively applied in various fields such as business activities, scientific computation, and engineering design. As these applications have deepened over time, a substantial volume of valuable data has been generated. However, faced with such intricate historical data, without the application of effective mining algorithms, accessing and utilizing this information becomes challenging, posing a significant obstacle to extracting meaningful insights from these massive datasets.

Data mining is the result of prolonged research and development in database technology. Initially, various business data was stored in computer databases, evolving to enable querying and accessing databases, and further advancing to real-time traversal. Data mining has propelled database technology into a more advanced stage, enabling not only the querying and traversal of past data but also the discovery of potential connections between data, thereby fostering the generation of useful information.

Association rule mining [1], as one of the crucial research areas in data mining, primarily focuses on the associations between data and various other interesting patterns. Since the introduction of the Apriori method by R.Agrawal et al. [2], numerous researchers have conducted extensive studies on the mining of frequent itemsets.

Algorithms based on frequent itemsets include the Apriori algorithm and FP-growth algorithm [3], [4]. The Apriori algorithm has inherent and insurmountable drawbacks, requiring repetitive database scans leading to significant I/O overhead and generating numerous candidate itemsets with substantial computational requirements.

The FP-growth algorithm is fundamentally different from the Apriori algorithm and represents another classic algorithm for mining frequent patterns [5]. In comparison to the Apriori algorithm [6], FP-growth does not need to generate a large number of candidate itemsets and requires only two scans of the database, resulting in improved algorithmic efficiency. Performance analysis indicates the effectiveness of the FP-growth algorithm in mining both long and short frequent patterns, achieving mining speeds approximately an order of magnitude faster than the Apriori algorithm.

The primary motivations and contributions of this paper are as follows:

- In pursuit of data mining, an in-depth investigation using the FP-Growth algorithm was conducted, leading to the exploration and extraction of data from the dataset.
- The existing FP-Growth algorithm's shortcomings were analyzed. Improvements were achieved by altering the algorithm's treatment of item processing order and item projection direction. Two approaches, namely "backward processing, forward projection" and "forward processing, backward projection," were employed to enhance the FP-Growth algorithm, resulting in an Improved FP-Growth (ImFP-Growth).
- Through experimental comparisons between the conventional FP-Growth and the enhanced ImFP-Growth, it was demonstrated that ImFP-Growth attains higher efficiency and significantly improves performance in the target dataset.

The rest of the paper is organized as follows. Section II provides a review of relevant work on FP-Growth algorithm. In Section III, the basic principles and algorithmic framework of the FP-Growth algorithm are presented, along with an analysis of its limitations and the reasons behind these shortcomings. Section IV introduces two distinct variations of the ImFP-Growth algorithm, elucidating the basic concepts and framework of each. Section V provides a detailed exposition of the experimental procedures conducted on the target dataset, comparing the algorithmic performance. Finally, Section VI concludes this paper.

II. Related Works

Frequent itemset mining has long been a focal research issue in the field of data mining, with numerous works and achievements. Initially applied to discover association rules in databases, the Apriori algorithm was first proposed by R.Agrawal et al. [2]. Association rules reflect interesting connections between itemsets within a large dataset. Identifying all frequent itemsets is crucial for association rule mining algorithms. Given the substantial amount of data processed by the Apriori algorithm, the efficiency of the algorithm becomes paramount. Many researchers have optimized the algorithm's join and prune processes, resulting in various variants.

Mannila.H et al. introduced an improved algorithm called AprioriTid [7]. This algorithm, except for the first step, does not require scanning the entire transaction database when calculating the support for candidate frequent itemsets. It posits that if a transaction does not contain any of the candidate frequent k-itemsets, there is no need to include any elements related to that transaction during scanning.

Hannu.T et al. proposed the Aprioripro algorithm [8], which shares the basic idea of the AprioriTid algorithm in reducing scans for frequent items in the database. The Aprioripro algorithm differs by adding an attribute to the original database, reducing frequent scans for certain transactions through the attribute's values.

Combining Apriori and AprioriTid, R.Srikant et al. presented a hybrid mining algorithm, AprioriHybrid [9]. The fundamental idea is to use the Apriori algorithm in the early stages of scanning and switch to the AprioriTid algorithm when the number of records in the candidate itemset is small enough to fit into memory.

Although these algorithms are more efficient in mining than the Apriori algorithm, they share the same nature, generating a large number of candidate frequent itemsets during the mining process.

In contrast to the Apriori algorithm, the FP-Growth algorithm [4] employs a depth-first strategy for mining frequent itemsets. The FP-Growth algorithm compresses the database into a Frequent Pattern Tree (FP-Tree) and then generates all frequent itemsets based on the FP-Tree through a frequent itemset growth approach. The drawback of the FP-Growth algorithm lies in the recursive construction of numerous conditional FP-Trees when dealing with many long and complex itemsets, incurring high time and space costs. Consequently, scholars have proposed improvements to the FP-Growth algorithm.

The H-mine algorithm [10] utilizes arrays instead of trees to represent itemsets, providing better space efficiency and search efficiency in sparse databases. By predicting space utilization rates, it creates conditions for mining large databases (after compression) that cannot entirely fit into memory. However, H-mine degenerates into the FP-Growth algorithm when handling dense databases.

While these methods have improved the performance of the FP-Growth algorithm to varying degrees, it is crucial to note that most approaches only record information about frequent items. As a result, the FP-Tree model is a specialized model, limiting its application scope.

III. FP-growth algorithm

A. Basic Principles

The FP-Growth algorithm requires only two scans of the database and can mine all frequent itemsets without generating any candidate itemsets. It employs a "divide and conquer" strategy as follows: first, it compresses the database containing the frequent itemsets into an FP-Tree (Frequent Pattern Tree), while retaining the association information of itemsets; then, it divides this compressed database into a set of conditional databases (a special type of projected database), with each conditional database associated with one (or a group of) frequent itemset(s). The algorithm then individually mines each conditional database in a similar manner, thereby generating all frequent itemsets.

$\psi_X^{[t,r]}(k)$ denotes the CFO for the k-th frame at t-th antenna and r-th antenna between Alice (Eve) and Bob, modeled as

$$\psi_X^{[t,r]}(k) = \psi_{Xc}^{[t,r]} + \psi_{Xd}^{[t,r]}(k), X \in \{A, E\}, \tag{1}$$

where $\psi_{Xc}^{[t,r]}$ is the constant component representing oscillator mismatch and $\psi_{Xd}^{[t,r]}(k)$ is the variable component representing Doppler frequency shift.

Moreover, $\psi_{Xd}^{[t,r]}(k)$ is modeled as an autoregressive random process and represented using a first-order Markov process [**?**] as

$$\psi_{Xd}^{[t,r]}(k+1) = \beta\psi_{Xd}^{[t,r]}(k) + \sqrt{1-\beta^2}\psi_{Xn}^{[t,r]}(k), \tag{2}$$

where β is the temporal correlation coefficient of the Doppler shift between two consecutive frames, $\psi_{Xn}^{[t,r]}(k)$ is the random variation of CFO on k-th frame which is uncorrelated to $\psi_X^{[t,r]}(k)$.

The CFO at $k+1$-th frame can be expressed as

$$\psi_X^{[t,r]}(k+1) = (1-\beta)\psi_{Xc}^{[t,r]} + \beta\psi_X^{[t,r]}(k) + \sqrt{1-\beta^2}\psi_{Xn}^{[t,r]}(k). \tag{3}$$

B. Algorithmic Framework

The FP-Growth algorithm generates all frequent itemsets in a depth-first, recursive manner, transforming the problem of discovering long frequent itemsets into recursively discovering short itemsets and then connecting them with suffixes to obtain long itemsets, incrementally growing one item at a time. The algorithm generates all frequent itemsets rather than "maximal frequent itemsets." That is, once a frequent itemset is generated, all its subsets (which are also frequent itemsets) are also generated [11]. The main steps of the FP-Growth algorithm are presented in Algorithm 1.

Algorithm 1 FP_Growth Algorithm

```
Data: FP_Tree, $\alpha$
if FP_Tree contains only a single path $P$ then
    foreach combination $\gamma$ of nodes in path $P$ do
        Generate frequent itemset: $\beta = \alpha \cup \gamma$, and $\beta$.support
          = minimum support of nodes in $\gamma$
else
    for check each item $a_i$ in FP-Tree header table from the
      end to the beginning (i=n to 1) do
        Generate frequent itemset: $\beta = a_i \cup \alpha$, and $\beta$.support
          = $a_i$.support  Construct the conditional pattern base
          for $\beta$, then construct the conditional FP-Tree for $\beta$:
          Tree$\beta$  if Tree$\beta$ is not empty then
            Call FP_Growth (Tree$\beta$, $\beta$) ; // Recursively
              call itself for further growth
```

C. Limitations of FP-Growth Algorithm

The FP-Growth algorithm employs a recursive self-call method to address the subproblems obtained through partitioning [5]. Consequently, whenever resolving a subproblem, a new conditional FP-Tree must be constructed, and this process is carried out recursively. Thus, until a subproblem is completely resolved, all the generated conditional FP-Trees must be entirely preserved in the stack, even if some of the preceding conditional FP-Trees are no longer needed later.

According to statistics, the number of conditional FP-Trees generated by the FP-Growth algorithm is of the same order of magnitude as the number of frequent itemsets it produces. As an increasing number of recursively generated conditional FP-Trees fill up memory, the mining process may become impractical to continue in memory. This directly leads to a decrease in the algorithm's execution speed, and the establishment of these conditional FP-Trees also incurs a considerable amount of time, affecting algorithm efficiency.

The root cause of these shortcomings lies in the fact that FP-Growth algorithm uses a "forward processing, forward projection" approach. If the subsequent extracted projection information continues to be stored in the original FP-Tree (without generating new conditional FP-Trees to save it), conflicts arise. In other words, in the FP-Growth algorithm, if newly acquired projection information is to share the storage space of the original tree, it will inevitably overwrite information that will be needed later. Therefore, it becomes necessary to produce separate conditional FP-Trees to preserve this information. Since the original algorithm is executed recursively, the processing is consistently "forward processing, forward projection," inevitably leading to the generation of a series of conditional FP-Trees.

IV. ImFP-Growth algorithm

Algorithmic improvements can be achieved by either adjusting the original algorithm's handling sequence of items or modifying its projection direction for items. We refer to the resulting algorithms, which involve sequential adjustments on the original algorithm without the need to generate conditional FP-Trees for frequent itemset growth, as ImFP-Growth (Improved FP-Growth Algorithm). It encompasses two distinct implementation approaches.

A. ImFP-Growth 1: Backward Processing and Forward Projection

The adjustment to the original algorithm involves modifying the handling sequence of items, transforming it into a "backward processing, forward projection" approach. Adopting this scheme, when processing a particular item, those items following it remain unprocessed, while those preceding it have already been processed. Furthermore, since it involves forward projection, the projection information can directly overwrite the information of the preceding items. This is because these items have been processed, and their information is no longer required. Consequently, there is no need to generate new conditional FP-Trees to store this information. By consistently generating frequent itemsets according to the order specified in Scheme 1, all subsequent information can be stored in the original tree, allowing the entire process to be conducted within the original tree.

The main steps of the ImFP-Growth 1 are presented in Algorithm 2.

B. ImFP-Growth 2: Forward Processing and Backward Projection

The adjustment to the original algorithm involves modifying the projection direction of items, transforming it into a "forward processing, backward projection" approach. Adopting this scheme, when processing a particular item, those items preceding it remain unprocessed, while those following it have already been processed. Moreover, since it involves backward projection, the projection information can directly overwrite the information of the subsequent items, as they have already been processed and no longer require this information. Similarly, if frequent itemsets are consistently generated according to the order specified in Scheme 2, all operations can be performed within the original tree without the need to generate new conditional FP-Trees.

The main steps of the ImFP-Growth 2 are presented in Algorithm 3.

V. Experiments and Results

A. Experimental Environment and Dataset

Conducting simulation experiments, this research employs the FP-Growth algorithm and the ImFP-Growth algorithm for data mining, with a focus on analyzing the performance disparities among these three algorithms. The experimental setup includes an Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 2.21 GHz, 16.0 GB RAM, a 64-bit operating system, x64-based processor, and Windows 10 operating system. Both the FP-Growth and ImFP-Growth algorithms are implemented in Python, using JetBrains PyCharm Community Edition 2019.1.3 x64. The dataset utilized in this experiment is the

Algorithm 2 ImFP-Growth 1: Backward processing and forward projection

```
Data: Transaction database, Minimum support
Result: All frequent itemsets
Function FP_Growth(TDB, Min_Support)
    Build FP-Tree for the transaction database
    Mining_Backward(FP-Tree) for Process each item $a_i$ from FP-Tree header table from front to back (i=1 to n) do
        Generate frequent 1-itemset: $P = \{a_i\}$, and $P$.support = $a_i$.support
        if There are preceding items to $a_i$ then
            Projection_Forward(FP-Tree, $a_i$)
            Growth_Forward(FP-Tree, P, i)
    Growth_Forward(FP-Tree, P, i) for Check each item $a_j$ from FP-Tree header table from front to back (j=1 to i-1) do
        if $a_j$.support $\geq$ Min_Support then
            Generate frequent itemset: $P = P \cup \{a_j\}$, and $P$.support = $a_j$.support
            if There are preceding frequent items to $a_j$ then
                Projection_Forward(FP-Tree, $a_j$)
                Growth_Forward(FP-Tree, P, j)
    Projection_Forward(FP-Tree, $a_i$) for Initialize each item $a_j$ preceding $a_i$ in FP-Tree header table (j=1 to i-1) do
        $a_j$.support = 0, and clear the node chain for $a_j$
        for Each node $node_{a_i}$ on the node chain for $a_i$ in FP-Tree do
            for Each node $node_{a_j}$ on the prefix path of $node_{a_i}$ (j=1 to i-1) do
                if $node_{a_j}$ is not on the node chain for $a_j$ then
                    $node_{a_j}$.support = $node_{a_i}$.support
                    $a_j$.support = $a_j$.support + $node_{a_i}$.support
                    Attach $node_{a_j}$ to the node chain for $a_j$
                else
                    $node_{a_j}$.support = $node_{a_j}$.support + $node_{a_i}$.support
                    $a_j$.support = $a_j$.support + $node_{a_i}$.support
```

array-data.csv dataset provided in class, serving as the target dataset for data mining experiments and algorithm performance evaluations.

B. Results of FP-Growth

For the target dataset, the results of data mining using the FP-Growth algorithm with a minimum support threshold of 5 are shown in Figure 1. Due to space constraints, only the results of the first 15 frequent itemsets will be displayed. Please refer to the submitted attachment for all the results.

C. Results of ImFP-Growth 1

For the target dataset, the results of data mining using the ImFP-Growth 1 algorithm with a minimum support threshold of 5 are shown in Figure 2. Due to space constraints, only the results of the first 15 frequent itemsets will be displayed. Please refer to the submitted attachment for all the results.

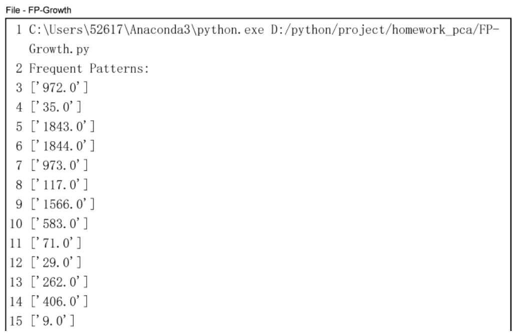
Algorithm 3 ImFP-Growth 2: Forward processing and backward projection

```
Data: FP_Tree, $\alpha$, Minimum support
Result: All frequent itemsets
if FP_Tree contains only a single path $P$ then
    foreach combination $\gamma$ of nodes in path $P$ do
        Generate frequent itemset: $\beta = \alpha \cup \gamma$, and $\beta$.support = minimum support of nodes in $\gamma$
else
    for check each item $a_i$ in FP-Tree header table from the end to the beginning (i=n to 1) do
        Generate frequent itemset: $\beta = a_i \cup \alpha$, and $\beta$.support = $a_i$.support
        Construct the conditional pattern base for $\beta$, then construct the conditional FP-Tree for $\beta$: Tree$\beta$
        if Tree$\beta$ is not empty then
            Call FP_Growth (Tree$\beta$, $\beta$) ; // Recursively call itself for further growth
for Process each item $a_i$ from FP-Tree header table from back to front (i=n to 1) do
    Generate frequent 1-itemset: $P = \{a_i\}$, and $P$.support = $a_i$.support
    if There are succeeding items to $a_i$ then
        Projection_Backward (FP-Tree, $a_i$)
        Growth_Backward (FP-Tree, $P$, i)
Growth_Backward (FP-Tree, $P$, i) for Check each item $a_j$ from FP-Tree header table after $a_i$ from back to front (j=n to i+1) do
    if $a_j$.support $\geq$ Minimum support then
        Generate frequent itemset: $P = P \cup \{a_j\}$, and $P$.support = $a_j$.support
        if There are succeeding frequent items to $a_j$ then
            Projection_Backward (FP-Tree, $a_j$)
            Growth_Backward (FP-Tree, $P$, j)
Projection_Backward (FP-Tree, $a_i$) for In FP-Tree header table, for each item $a_j$ after $a_i$ (j=i+1 to n) do
    $a_j$.support = 0, and clear the node chain for $a_j$
    for In FP-Tree, for each node $node_{a_i}$ on the node chain for $a_i$ do
        for For each node $node_{a_j}$ in the subtree rooted at $node_{a_i}$, perform depth-first traversal, for each traversed node $node_{a_j}$ (j=i+1 to n) do
            $a_j$.support = $a_j$.support + $node_{a_j}$.support
            Attach it to the node chain for $a_j$
```

the results of the first 15 frequent itemsets will be displayed. Please refer to the submitted attachment for all the results.

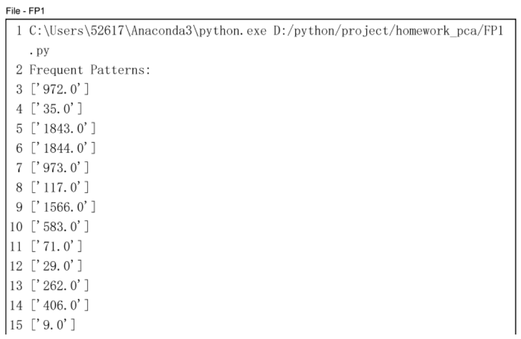
D. Results of ImFP-Growth 2

For the target dataset, the results of data mining using the ImFP-Growth 2 algorithm with a minimum support threshold of 5 are shown in Figure 3. Due to space constraints, only the results of the first 15 frequent itemsets will be displayed. Please refer to the submitted attachment for all the results.


File - FP-Growth

```
C:\Users\52617\Anaconda3\python.exe D:/python/project/homework_pca/FP-
Growth.py
Frequent Patterns:
['972.0']
['35.0']
['1843.0']
['1844.0']
['973.0']
['117.0']
['1566.0']
['583.0']
['71.0']
['29.0']
['262.0']
['406.0']
['9.0']
```

Fig. 1: First 15 Results of FP-Growth


File - FP1

```
C:\Users\52617\Anaconda3\python.exe D:/python/project/homework_pca/FP1
.py
Frequent Patterns:
['972.0']
['35.0']
['1843.0']
['1844.0']
['973.0']
['117.0']
['1566.0']
['583.0']
['71.0']
['29.0']
['262.0']
['406.0']
['9.0']
```

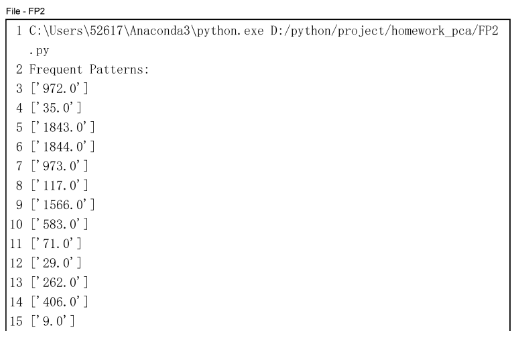
Fig. 2: First 15 Results of ImFP-Growth 1

E. Performance Comparison

From the experimental results, it can be seen that the algorithms mentioned in this article can correctly implement data mining and achieve frequent itemset results.

In order to compare the performance of the algorithms mentioned in this article, a large number of experiments were designed to make comparisons in terms of elapsed time and max memory [12].

As shown in Tables I, II, and III, when the minimum support thresholds are set to [0.01, 0.05], [0.1, 0.5], and [1, 5], the elapsed time and max memory of the three algorithms, ImFP-Growth 1, ImFP-Growth 2, and FP-Growth, are presented. Corresponding comparisons of the elapsed time and max memory of the three algorithms are illustrated in Figures 4, 5, and 6.


File - FP2

```
C:\Users\52617\Anaconda3\python.exe D:/python/project/homework_pca/FP2
.py
Frequent Patterns:
['972.0']
['35.0']
['1843.0']
['1844.0']
['973.0']
['117.0']
['1566.0']
['583.0']
['71.0']
['29.0']
['262.0']
['406.0']
['9.0']
```

Fig. 3: First 15 Results of ImFP-Growth 2

TABLE I: Elapsed Time and Max Memory at Minimum Support Threshold Ranges [0.01, 0.05]

Algorithm	Elapsed Time (seconds)				
	0.010	0.020	0.030	0.040	0.050
ImFP-Growth 1	292.68	285.05	282.00	286.58	291.16
ImFP-Growth 2	288.61	289.63	294.21	292.68	294.21
FP-Growth	965.60	959.50	977.81	979.33	985.44
Algorithm	Max Memory (MB)				
	0.010	0.020	0.030	0.040	0.050
ImFP-Growth 1	84.58	90.60	92.78	92.50	93.93
ImFP-Growth 2	83.54	90.60	92.78	92.50	93.93
FP-Growth	90.55	92.23	92.53	93.07	86.31

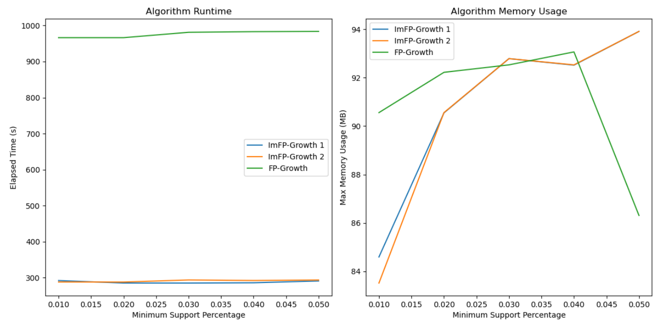


Fig. 4: Algorithm Comparison at Minimum Support Threshold Ranges [0.01, 0.05]

TABLE II: Elapsed Time and Max Memory at Minimum Support Threshold Ranges [0.1, 0.5]

Algorithm	Elapsed Time (seconds)				
	0.10	0.20	0.30	0.40	0.50
ImFP-Growth 1	313.61	294.41	303.17	290.46	293.55
ImFP-Growth 2	335.52	299.28	295.37	289.61	296.25
FP-Growth	993.48	1002.11	980.06	974.85	982.56
Algorithm	Max Memory (MB)				
	0.10	0.20	0.30	0.40	0.50
ImFP-Growth 1	84.39	90.60	85.44	85.04	86.01
ImFP-Growth 2	84.41	90.62	85.21	84.98	86.01
FP-Growth	90.62	84.94	85.04	85.18	85.76

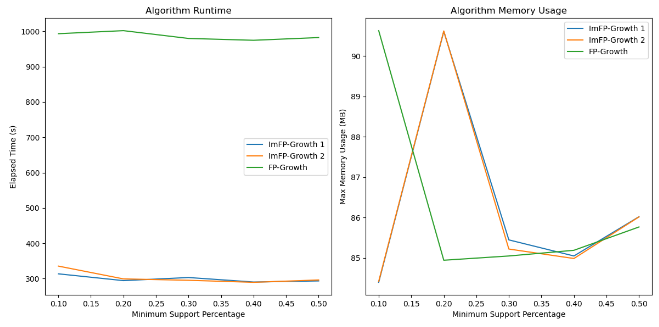


Fig. 5: Algorithm Comparison at Minimum Support Threshold Ranges [0.1, 0.5]

TABLE III: Elapsed Time and Max Memory at Minimum Support Threshold Ranges [1, 5]

Algorithm	Elapsed Time (seconds)				
	1	2	3	4	5
ImFP-Growth 1	295.79	294.21	292.12	291.13	298.28
ImFP-Growth 2	290.81	290.30	290.14	291.58	294.71
FP-Growth	978.23	296.48	295.18	300.81	301.62
Algorithm	**Max Memory (MB)**				
	1	2	3	4	5
ImFP-Growth 1	84.47	90.51	92.58	91.90	86.25
ImFP-Growth 2	84.49	90.53	91.82	85.53	86.23
FP-Growth	91.30	92.58	91.90	86.25	86.42

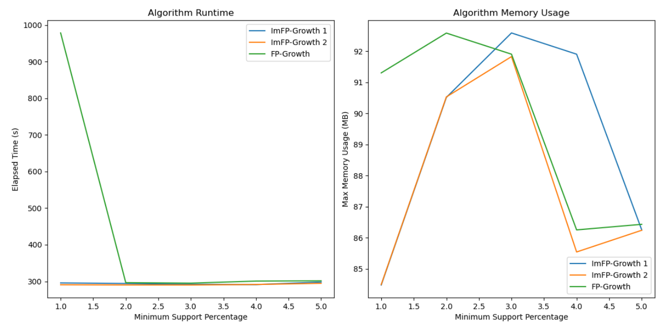


Fig. 6: Algorithm Comparison at Minimum Support Threshold Ranges [1, 5]

From the perspective of algorithm elapsed time, as observed in Table I, Table II and Table III, it can be seen that in different minimum support intervals, the ImFP-Growth algorithm outperforms the FP-Growth algorithm, with little difference in the execution time between the two ImFP-Growth algorithm schemes. As depicted in Figure 1, Figure 2, and Figure 3, it is evident that when the minimum support is relatively low, the advantage of the ImFP-Growth algorithm becomes more pronounced. The ImFP-Growth algorithm has achieved a significant improvement over the FP-Growth algorithm in terms of elapsed time.

From the perspective of algorithm max memory, although there is no consistent pattern in the changing trend of max memory for the three algorithms in Figure 1, Figure 2, and Figure 3 across different minimum support intervals, it can be observed from the data in Table I, Table II and Table III that there is little difference in the max memory corresponding to the three schemes. While the ImFP-Growth algorithm has not significantly improved over the FP-Growth algorithm in terms of max memory, it has not led to a significant increase in max memory either, which may be related to the experimental operating environment. It is also noticeable that when the minimum support is relatively low, the two schemes of the ImFP-Growth algorithm require almost the same amount of max memory, and the trend of max memory for the two schemes of the ImFP-Growth algorithm is consistent across different value intervals.

VI. Conclusion

In this paper, an in-depth study of data mining is conducted through the utilization of the FP-Growth algorithm, implementing data mining on the target dataset. The existing deficiencies of the FP-Growth algorithm are analyzed, and two improved versions, ImFP-Growth, are proposed by altering the algorithm's processing order and projection direction for items. Extensive experiments demonstrate that the algorithms mentioned in this paper yield correct results. Moreover, in the target dataset, ImFP-Growth algorithm achieves higher efficiency with smaller minimum support, without causing an increase in max memory.

References

[1] R. Agrawal, T. Imieliński, and A. Swami, Mining association rules between sets of items in large databases, Proc. 1993 ACM SIGMOD Int. Conf. Manag. data - SIGMOD '93, no. May, pp. 207–216, 1993.
[2] R.Agrawal and R.Srikant. Fast algorithms for mining association rules[C]. Proceedings of the 20th International Conference on Very Large Database, 1994:487-499.
[3] M.J.Zaki, Scalable algorithms for association mining, in IEEE Transactions on Knowledge and Data Engineering, vol. 12, no. 3, pp. 372-390, May-June 2000, doi: 10.1109/69.846291.
[4] Jiawei Han, Jian Pei, Yiwen Yin. Mining frequent patterns without candidate generation[C]. Proc. ACM-SIGMOD'2000, Dallas, TX, May 2000: 1-12.
[5] Z.Le, Y.Z.Zhi, C.Jin. Balanced Parallel FP-Growth with MapReduce[C]. Proceedings of the IEEE Youth Conference on Information Computing andTelecom, beijing, 2010:243-246
[6] W. Zhang, H. Liao and N. Zhao, Research on the FP Growth Algorithm about Association Rule Mining, 2008 International Seminar on Business and Information Management, Wuhan, China, 2008, pp. 315-318, doi: 10.1109/ISBIM.2008.177.
[7] Mannila H, Toivonen H, Inkeri Verkamo A. Efficient algorithms for discovery association rules[C]. In Proceedings of AAAI Workshop on Knowledge Discovery in Database 1994. 181-192.
[8] Hannu Toivonen, Mika Klemettinen, Pirjo Ronkaine. Pruning and grouping discovered association Rules[C]. In Workshop Notes of the ECML-95 Workshop on Statistics, Machine Learning, and Knowledge Discovery in Databases. Berlin: Springer-Verlag, 1995: 47-52.
[9] R.Srikant and R.Agrawal. Mining quantitative association rules in large relational tables[C]. SIGMOD Conefrenee. 1996: 1-12
[10] J.Han, H.Lu, S.Nishio, S.Tang, and D.Yang. Hmine: Hyper-structure, Mining of frequent patterns in large databases[C]. In Proceedings of IEEE International Conference on Data Mining, 2001: 441-448.
[11] Hasan M M , Zaman Mishu S .An Adaptive Method for Mining Frequent Itemsets Based on Apriori And FP Growth Algorithm[C]//2018:1-4.DOI:10.1109/IC4ME2.2018.8465499.
[12] J. Yan and W. Qi, Research based on aprior and FP-growth algorithm, Computer system application, vol. 22, no. 5, pp. 122-125, 2013.