Journal of Database Management
Volume 34 • Issue 1

Frequent Itemset Mining Algorithm Based on Linear Table

Jun Lu, Heilongjiang University, China
Wenhe Xu, Heilongjiang University, China*
Kailong Zhou, Heilongjiang University, China
Zhicong Guo, Heilongjiang University, China

ABSTRACT

Aiming at the speed of frequent itemset mining, a new frequent itemset mining algorithm based on a linear table is proposed. The linear table can store more shared information and reduce the number of scans to the original dataset. Furthermore, operations such as pruning and grouping are also used to optimize the algorithm. For different datasets, the algorithm shows different mining speeds. (1) In sparse datasets, the algorithm achieves an average 45% improvement in mining speed over the bit combination algorithm, and a 2-3 times improvement for the classic FP-growth algorithm. (2) In dense datasets, the average improvement over the classic FP-growth algorithm is 50-70%. For the bit combination algorithm, there are dozens of times of improvement. In fact, the algorithm that integrates bit combinations with bitwise AND operation can effectively avoid recursive operations and it is beneficial to the parallelization. Further analysis shows that the linear table is easy to split to facilitate the data batch mining processing.

KEYWORDS

Apriori, Bit Combinations, Data Mining, FP-growth, Grouping, Recursive, Support, Pruning

INTRODUCTION

Data mining (Witten & Frank, 2002) refers to the process of searching for hidden information from a large amount of data through algorithms. In recent years, data mining has attracted great attention in the information industry. The main reason is that there is a large amount of data, and it is urgent to transform this data into useful information and knowledge.

Frequent itemset mining is an important basis for research in data mining. It is the premise of feature selection, cluster analysis, and association rule mining (Fiori et al., 2014). By mining frequent itemsets, the combination of frequent items that usually appeared in the original dataset can be obtained. It can also provide some support for possible decision-making. Frequent itemset mining has a wide range of applications, such as shopping basket data analysis (Wang, 2014), cross-shopping (Nafie Ali & Mohamed Hamed, 2018), traffic accident analysis (Hidayanto et al., 2017), and network intrusion detection (Liao et al., 2012). At the same time, frequent itemset mining is also applied in the field of life science.

At present, there are several different mainstream methods for frequent itemset mining. The principle of the Apriori-like algorithm is that all nonempty subsets of frequent itemsets must also be

DOI: 10.4018/JDM.318450 *Corresponding Author

frequent. The FP-growth-like algorithm uses a novel tree structure. In the FP-growth-like algorithm, to share the data, a frequent pattern tree is constructed according to the transaction database. Subsequently, the frequent itemsets are mined from the frequent pattern tree. Only two scan times on the original dataset are needed. Compared with the Apriori-like algorithm, the FP-growth-like algorithm improves the calculation speed and reduces the running time. In addition, some researchers use bitmaps to represent transactions, and some frequent itemset mining methods based on bit operations have been proposed to further improve the operation speed and the efficiency of memory usage.

RELATED WORK

The Apriori-like algorithm is a kind of frequent itemset mining algorithm with a relatively simple structure, and the mining process adopts the method of breadth-first traversal. The IAA (Wu et al., 2009) uses a count-based and record-generated method to improve the performance of the Apriori algorithm. Yuan et al. (2017) further use an overlapping strategy to compute support for achieving high efficiency. The FP-growth algorithm (Han et al., 2000) constructs an FP-tree structure and uses depth-first traversal for mining. A new DPT algorithm (Qu et al., 2020) is proposed, which only uses a dynamic prefix tree structure and can directly output a prefix tree representing all frequent itemsets, thus avoiding the high cost of building many prefix trees. The negFIN algorithm (Aryabarzan et al., 2018) adopts the bitwise operation to extract the NegNodesets structure of the itemset and uses a set enumeration tree to generate frequent itemsets, which is pruned by the promotion method. FDM is a new algorithm based on FP-tree and DIFFset (Gatuha & Jiang, 2017) data structures, which can mine long and short patterns from dense and sparse datasets. The SS-FIM algorithm (Djenouri et al., 2017) performs a single scan of the transactional database to extract frequent itemsets, and the algorithm is less sensitive to the changes in the minimum support threshold set by the user. The dFIN algorithm (Deng, 2016) represents the itemsets through the DiffNodeset structure. The algorithm uses a hybrid search strategy to find frequent itemsets on the set enumeration tree and directly enumerates frequent itemsets without generating candidate itemsets under certain conditions. The PrePost (Deng et al., 2012) and PrePost+ (Deng et al., 2015) use a vertical N-list structure to store the key information of frequent itemsets, which calculates the support through the intersection of N-lists. In particular, the PrePost+ algorithm adopts an effective pruning strategy called Children-Parent Equivalence pruning, which reduces the search space. The Nodeset (Deng & Lv, 2014) is a more efficient data structure, it only needs pre-order or post-order encoding of each node, which makes it save half the memory compared with the N-list.

The SHFIM is a spark-based three-stage algorithm for hybrid frequent itemset mining (Al-Bana et al., 2022), which utilizes horizontal and vertical layout difference sets to track differences among transaction IDs. Liu et al. (2022) adopted an indexed prefix closed itemset tree to compress all closed itemsets and proposed a new strategy to prune and update the search space of closed itemsets, so as to quickly mine closed itemsets. NEclatClosed (Aryabarzan & Minaei-Bidgoli, 2021) is a vertical algorithm for fast mining of frequent closed itemsets. The algorithm applies the concepts and techniques of vertical databases in the mining process. Vo et al. (2012) proposed a frequent closed itemset mining algorithm based on dynamic bit vector (DBV) and introduced a fast way to compute the intersection between two DBVs. Mining frequent weighted itemsets (Bui et al., 2021; Li et al., 2021; Zahra et al., 2020) considers both the minimum support threshold and the weight coefficients of items, which makes extracting knowledge more meaningful. The FPMSIM algorithm (Xun et al., 2021) is an incremental frequent itemset mining algorithm, which uses similarity estimation in the process of maintaining incremental updates, avoiding severe dataset rescanning and tree structure adjustment overhead. In addition, the AFARTICA algorithm (Paladhi et al., 2019) proposed a pruning technique based on artificial cell division (ACD) to solve the problem of multiple search spaces. Lessanibahri et al. (2020) proposed a pruning method named LengthSort, which prunes items and transactions before building a frequent pattern tree structure, thereby reducing the search space.

The above algorithms evolve from the classic Apriori algorithm and FP-growth algorithm. The Apriori algorithm has obvious advantages: it is straightforward, easy to understand, and does not involve the use of complex data structures. However, when candidate sets are generated, it fails to exclude elements that should not participate in the combination in time, and it needs to scan the original database repeatedly when calculating support. As a result, the disk has too many I/O cycles, and the efficiency is relatively low.

The FP-growth algorithm overcomes some shortcomings of the Apriori algorithm. However, when the database is very large, the algorithm will generate a very complex and large tree. It is difficult to deal with batch data by organizing the data with the tree structure. Furthermore, the algorithm uses recursive operations so that it is not easy to parallelize.

In addition, there are some frequent itemset mining algorithms based on bit operations. Examples include the frequent closed itemset mining algorithm based on bit operations (Jia-Li et al., 2013), the SECRET algorithm (Dobra & Gehrke, 2002), and the frequent itemset mining algorithm based on bit combinations (Lu et al., 2019). Among them, the frequent closed itemset mining algorithm based on bit operations needs to transform the dataset into a Boolean matrix and scan the dataset only once. Support is calculated by bit operation, and auxiliary information is stored by the matrix and array to reduce time and space consumption. The pruning strategy is used to further reduce the mining time when the frequent closed itemsets are generated by depth-first search without checking the supersets or subsets. However, this algorithm takes into account the two types of information of attribute distance and relationship strength so that the time complexity of the clustering process reaches $O(n^2)$, which can be reduced to O(nlogn) after using the index. The efficiency of the algorithm needs to be further improved.

The SECRET algorithm proposes a new sequential pattern mining algorithm using a depth-first search strategy. This strategy combines a depth-first traversal search space with an effective pruning mechanism. It also combines the vertical bitmap representation of the database with the effective support count. Although the mining speed has been greatly improved, the algorithm is a nonprecise mining algorithm that cannot incorporate speed and accuracy at the same time, which is its drawback.

The frequent itemset mining algorithm based on bit combination adopts binary bits to represent the itemset. The binary digit representing the itemset is gradually added to 1 to judge, and then the frequent itemsets are found by bitwise AND calculation with each transaction in the dataset. However, the algorithm requires multiple scans of the original dataset, which affects the algorithm's mining efficiency.

Motivations and Contributions

Aiming at the shortcomings of the existing algorithms, a frequent itemset mining algorithm based on a linear table is proposed. This method is an accurate mining algorithm. This is unlike the Apriori algorithm, which generates too many candidate itemsets. It also avoids the recursive operation of the FP-growth algorithm in the mining process, making the algorithm more conducive to parallelism. At the same time, it makes full use of shared information such as the FP-growth algorithm, thereby improving computational efficiency. The linear table structure adopted by the new algorithm is conducive to data splitting, which facilitates big data mining.

The contribution of this paper mainly includes the following 3 parts:

(1) A frequent itemset mining algorithm based on a linear table is proposed. The algorithm can avoid the memory consumption caused by recursive operations in the mining process and is convenient for parallel processing in the future.
(2) Construct a linear table by combining binary data. The linear table is more conducive to data splitting than the tree structure, which is convenient for frequent itemset mining for large datasets in the future.
(3) Pruning and grouping strategies are used to optimize the frequent itemset mining algorithm based on a linear table, which greatly improves the mining speed.

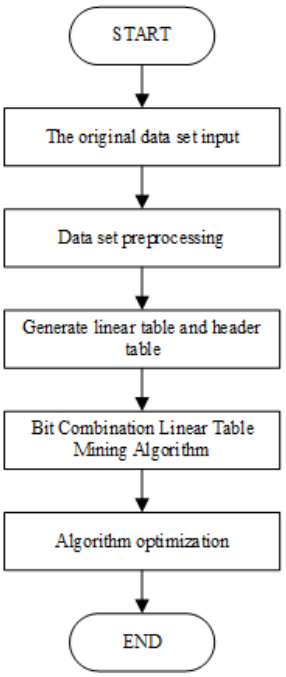
METHOD

The frequent itemset mining algorithm based on a linear table stores the dataset in a linear table through grouping optimization. When mining frequent itemsets, there is no need to rescan the original database, and only the existing linear table is scanned. At the same time, in the process of frequent itemset mining, each transaction item information is converted into binary numbers, and with pruning and grouping strategies, it is judged whether it is a frequent itemset by the bitwise AND operation, which improves the efficiency of mining.

For the convenience of description, the structure is used to represent each node in the linear table. The related information is represented in the form $\left(d_0, d_1, d_2, d_3, d_4, d_5, d_6\right)$. d_0 represents the name of the item. d_1 represents the index of the child node of the current item d_0 in the linear table. d_2 represents the index of the sibling node of the current item d_0 in the linear table. That is, d_2 and d_0 have the same parent node. d_3 represents the index of the parent node of the current item node d_0 in the linear table. d_4 represents the index of the last occurrence of the current item d_0 in the linear table. d_4 acts as a pointer, which can quickly find the position of the last occurrence of the item and speed up mining. For example, an item named x can be represented in a linear table as $\left(x, d_1, d_2, d_3, d_4, d_5, d_6\right)$. Among them, d_4 indicates that the position of the previous occurrence of Item x is in position d_4 of the linear table. The function of d_5 is to count. It mainly records the specific support of transactions. d_6 represents the binary number representation of the substring ending with the data Item x in the current transaction. Binary 0 indicates that the item does not exist in this location, and binary 1 indicates that the item exists in this location.

The overall design of the algorithm in this paper is mainly composed of three parts, and the framework of the algorithm is shown in Figure 1.

Figure 1. Overall design framework of the algorithm



The first step is the preprocessing of the original transaction dataset.

First, the original dataset is scanned once to obtain the support count of the one-itemset. Then, according to the preset minimum support, the items less than the minimum support in the one-itemset are filtered out. For the convenience of subsequent processing, items that are greater than or equal to the minimum support are sorted in descending order according to global support.

The second step is to construct a linear table and a head table.

According to the preprocessed dataset obtained in the first step, the original linear table is constructed. The subscript of the linear table represents the serial number value of each item and serves as the node number of the parent node or sibling node of one node. To improve the efficiency of linear table construction and algorithm mining, a header table will be generated synchronously in the process of constructing a linear table. The items in the header table are sorted in descending order by frequency. Through this header table, the target item can be quickly found when inserting items or frequent itemset mining.

The third step is to mine frequent itemsets.

After constructing the linear table and the header table, the next step is to mine frequent itemsets. The mining process adopts the binary bit combination method. The binary combination number Data will add 1 from all 0 to all 1. Every time 1 is added, a binary data BIT_ARRAY corresponding to a new combination item is obtained, and then the bitwise AND operation of the BIT_ARRAY and d_6 of the corresponding node in the linear table is performed. If the result is the same as the BIT_ARRAY, the support is summed. Next, find the next node that needs to be judged according to d_4. The above operation is repeated until d_4 is equal to 0. Then, the final support of the combination item is obtained. By comparison with the preset support, it can be judged whether the combination item is frequent. In the mining process, with multiple optimization strategies, the speed of frequent itemset mining can be accelerated.

Algorithm Description

Preprocessing

The frequent itemset mining algorithm based on a linear table is proposed in this paper. To facilitate understanding, a small-scale example of the original dataset is adopted. Table 1 shows the six raw data to be mined. The minimum support is set to 3.

After the first scan, the support count of the one-itemset can be obtained. Then, according to the preset minimum support, the items less than the minimum support in the one-itemset are filtered out. The filtered result is shown with Items in the transaction in Table 2. The items are sorted according to global support. The order of the filtered data items is z, x, y, t, s, and r. The result is shown in the last column in Table 2.

Table 1. The original dataset

TID	Items in the transaction
001	r,z,h,j,p
002	z,y,x,w,u,v,t,s
003	z
004	r,x,n,o,s
005	y,r,x,z,q,t,p
006	y,z,x,e,q,s,t,m

Table 2. Dataset after the first scan

TID	Items in the transaction	Filtered and sorted transactions
001	r,z,h,j,p	z,r
002	z,y,x,w,u,v,t,s	z,x,y,t,s
003	z	z
004	r,x,n,o,s	x,s,r
005	y,r,x,z,q,t,p	z,x,y,t,r
006	y,z,x,e,q,s,t,m	z,x,y,t,s

Building Linear Table

A complete linear table will be constructed from the information in Table 2.

1. T001 is z and r, so z and r need to be inserted into the linear table.

After inserting z, it can be determined that the child node index of z is 0, the sibling node index is 0, the parent node index is 0, and the index of the previous item is the same as the current item is 0, and the frequency is 1. At this time, d_1, d_2, d_3, d_4 of the first term z are all 0. d_5 is 1. According to the sorting of items after filtering, the binary data corresponding to Item z in T001 is 100000. That is, d_6 is 100000.

After inserting r, the child index of r is 0, the sibling node index is 0, the parent node index is 1, the index of the previous item is the same as the current item is 0, and the frequency is 1. After inserting r, the child node index of z needs to change synchronously. That is, d_1 of z is changed to 2. Similarly, according to the order of filtered items, d_6 of Item r is represented as the binary number form 10001 of the current transaction substring zr. The linear table after inserting T001 is shown in Table 3.

Table 3. Linear table after inserting T001

0	Φ(name, child, sibling, parent, point, frequency, binary number)
1	(z,2,0,0,0,1,100000)
2	(r,0,0,1,0,1,100001)

2. T002 is z, x, y, t, s, so these items need to be inserted into the linear table.

Since the term z has already appeared in T001, the first term z in T002 can share a node with the existing z term. That is, the d_5 frequency in the number 1 item (z,2,0,0,0,1,100000) in the linear table needs to be increased by one, and item information becomes (z,2,0,0,0,2,100000).

Insert the second term x of T002. It determines that the second term r [1] x is in the linear table, and the sibling node of r is 0, so a new Item x needs to be inserted into the linear table. That is, (x,0,0,1,0,1,110000), the d_1 child node is still uncertain at this time, so the default is 0. The d_2 sibling node is also unknown at this time. It is also 0 by default. d_3 is the parent node, the parent node of Item x is z, and the index value of Item z is 1, so the parent node d_3 of Item x is 1. d_4 is the previous

index whose name is the same as the current item name. At this time, Item x has not appeared before in the linear table, so d_4 is also 0. d_5 is the frequency of the current item, and the current frequency is 1, so d_5 is 1. Because the parent node of Item x is z and d_6 represents the binary number of the substring zx, the binary number of Item x is 110000. After the values of d_1 to d_6 are determined, it is indicated that the term x is inserted, and the index value of Item x in the linear table is 3, which is the sibling node of the Item r. Therefore, the value of d_2 (sibling node) in (r, 0,0,1,0,1,100001) should be changed to 3 (index of Item x in linear table), and the rest information should remain unchanged, that is, (r,0,3,1,0,1,100001).

Insert Item y, the initial insertion information is (y,0,0,3,0,1,111000). Since Item y is a child of x, the d_1 (child node information) of term x is changed to 4, and the information in the third item in the linear table becomes (x, 4,0,1,0,1,110000).

Insert Item t, the initial insertion information is (t,0,0,4,0,1,111100). Since Item t is a child node of y, the d_1 (child node information) of Item y should be changed to 5, and the information in the fourth item in the linear table should be changed to (y, 5,0,3,0,1,111000).

For Item s, the initial insertion information is (s,0,0,5,0,1,111110). Since Item s is a child node of t, the d_1 (child node information) of Item t should be changed to 6 at this time, and the information in the fifth item in the linear table should be changed to (t,6,0,4,0,1,111100). Since there is no child node under Item s, the sixth item information in the linear table is unchanged and is still (s,0,0,5,0,1,111110). The linear table after inserting T002 is shown in Table 4.

Table 4. Linear table after inserting T002

0	Φ(name, child, sibling, parent, point, frequency, binary number)
1	(z,2,0,0,0,2,100000)
2	(r,0,3,1,0,1,100001)
3	(x,4,0,1,0,1,110000)
4	(y,5,0,3,0,1,111000)
5	(t,6,0,4,0,1,111100)
6	(s,0,0,5,0,1,111110)

3. Next, the insertion process of T003 (z), T004 (x, s, r), T005 (z, x, y, t, r), and T006 (z, x, y, t, s) is similar to the previous description process. The linear table after the final insertion is shown in Table 5.

The logical structure of a linear table is actually equivalent to a binary tree, which is also different from the FP-growth algorithm. The FP-tree is actually a multi-branch tree.

It is worth mentioning that in the process of building a linear table, a real-time updated header table must be generated. Based on the header table, the position of the target item in the linear table can be quickly found when the transaction data are inserted. The data item insertion process can be effectively accelerated. In this example, the generated header table is shown in Table 6.

The 0th item in the header table indicates the data item name. The first item stored in the header table is the index of each item that appears as the parent node for the first time. The purpose of this item is to quickly find the first item without having to traverse every item from the top of the linear table. The second item of information stored in the header table is the index of the position where the item currently appears. The purpose of this item is to store d_4 which indicates node information

Table 5. Final linear table

0	Φ(name, child, sibling, parent, point, frequency, binary number)
1	(z,2,7,0,0,5,100000)
2	(r,0,3,1,0,1,100001)
3	(x,4,0,1,0,3,110000)
4	(y,5,0,3,0,3,111000)
5	(t,6,0,4,0,3,111100)
6	(s,0,10,5,0,2,111110)
7	(x,8,0,0,3,1,010000)
8	(s,9,0,7,6,1,010010)
9	(r,0,0,8,2,1,010011)
10	(r,0,0,5,9,1,111101)

Table 6. Header table

Data item	Parent index	Indicator index
z	1	0
x	7	3®7
y	0	4
t	0	5
s	0	6®8
r	0	2®9®10

of the current item, which will be continuously updated as the inserted item changes. After adding the header table, the logical structure of the linear table is shown in Figure 2.

Notably, the method used to update the second item of information in the header table is relatively clever. For example, for Item r, the first occurrence of r in the linear table is at position 2. In (r,0,3,1,0,1,100001), the information of index 2 is first stored in the second position of Item r in the header table. With the insertion of each transaction to the linear table, Item r appears again. The second occurrence of r in the linear table is at position 9, and d_4 of (r,0,0,8,2,1,010011) indicates that the node will extract the previously stored index 2 from the header table. The second position of Item r in the header table needs to be updated, that is, 2 is overwritten with 9. In addition, 9 is overwritten with 10. In the process of continuously inserting transactions, the header table is constantly updated.

Frequent Itemset Mining

After the linear table and header table are constructed, frequent itemsets can be mined for all transaction information stored in the linear table. This algorithm adopts the minimum support 3 specified by the user, and after preprocessing and other optimization strategies, the dataset listed in Table 2 is obtained. Then, the items obtained after the processing are combined in a binary bit manner. It starts with binary 1 and adds 1 successively until the binary bits are all 1. The operation process is shown in Table 7.

The position of 1 in Table 7 indicates that the item exists, and the presence of multiple 1 means that it is an itemset. For example, the binary digit is 001101, representing a combination of Items y, t, and r.

Figure 2. Logical structure of adding header table

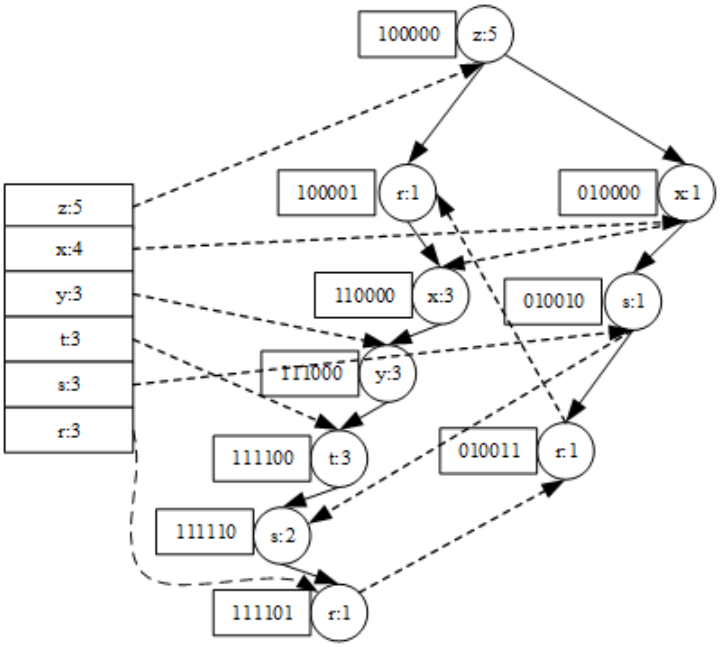


Table 7. Combinations of binary items

Data	z	x	y	t	s	r	Corresponding items and their combinations
1	0	0	0	0	0	1	r
2	0	0	0	0	1	0	s
3	0	0	0	0	1	1	s,r
4	0	0	0	1	0	0	t
……	……	……	……	……	……	……	……
63	1	1	1	1	1	1	z,x,y,t,s,r

When frequent itemset mining is performed. The number of binary item combinations in Table 7 will increase from all 0 to all 1. After adding 1 each time, a new item combination can be obtained. For example, the above combination of terms y, t, r has a binary combination digit of 001101. To determine whether this combination is a frequent itemset, the previously constructed linear table will be used. First, find the index of the last occurrence of Item r according to the header table, which is 10. Then, the d_6 binary digit with index 10 is 111101. Let 001101 and 111101 perform the bitwise AND operation, and the result is 001101. This proves that the item combination y, t, r exists in this transaction. Therefore, the d_5 frequency 1 with index 10 is added to the support of the combination. The next step is to find the position where the previous r appears according to the d_4 with index 10. By querying, d_4 indicates that the node is 9 at this time and the d_6 binary digit with index 9 is 010011. Do a bitwise AND operation between 001101 and 010011. The result is 000001, which is not equal to 001101. Then, it proves that the item combination y, t, r does not exist in this transaction, so the support of the combination will not increase. Continue to find the position where the previous r

appears according to the d_4 with index 9. It can be known by querying that d_4 indicates that the node is 2. The binary digit d_6 with index 2 is 100001, and a bitwise AND operation performs between 001101 and 100001. The result is 000001, and it is not equal to 001101. Then, it proves that the item combination y, t, r does not exist in this transaction, so the support of the item combination will not increase. Finally, the support of the item combination y, t, r is 1, which is lower than the minimum support specified by the user. Therefore, itemset y, t, r is infrequent. The logical structure of the frequent itemset mining process is shown in Figure 3.

Figure 3 shows that in the mining process, there is no need for child, parent, or sibling information. Therefore, for the convenience of viewing and analyzing, Figure 3 can be arranged in the form of Figure 4.

In Figure 4, based only on the head table and related nodes, the frequent itemsets ending with that element can be found. Finally, all the frequent itemsets will be found.

Figure 3. Logical structure of frequent itemset mining process

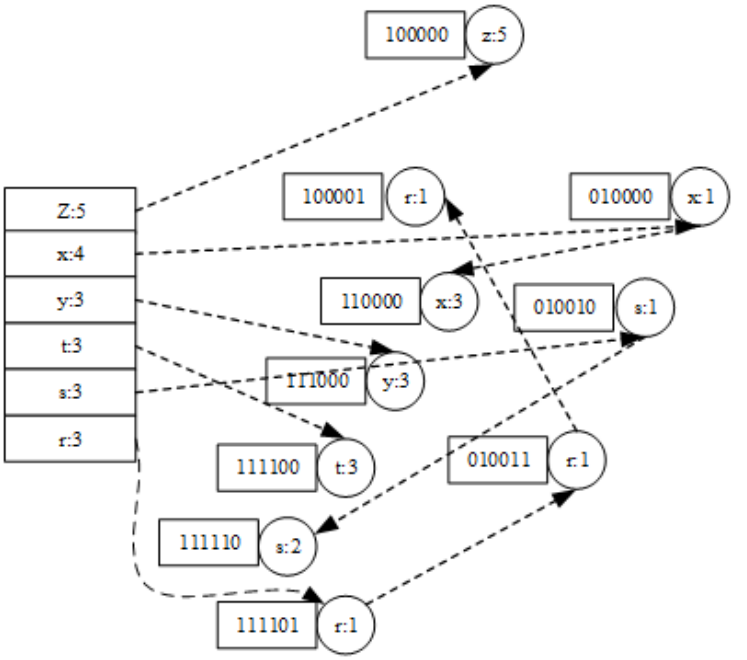
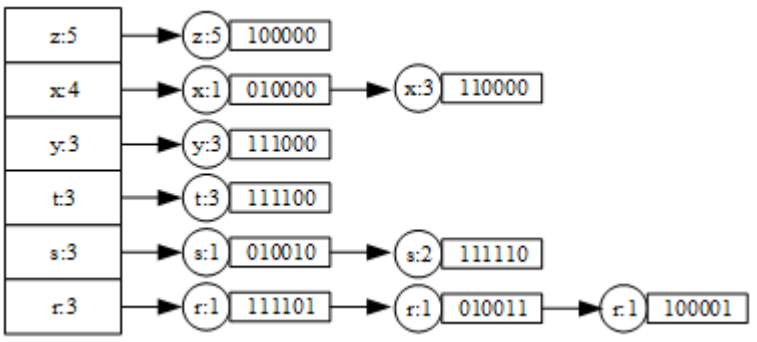


Figure 4. Mining logical structure



Complexity Analysis

After the dataset is preprocessed, let the number of rows be N, the average number of columns be L, and the number of frequent one-itemset be M. Then, the space complexity of constructing the linear table in this algorithm is O(N * L). The mining process is carried out on the linear table, and no additional space is allocated. That is, the complexity of this process is O(1), so the total space complexity is O(N*L). At the same time, the algorithm scans the dataset a finite number of times to construct a linear table, so the time complexity of this process is O(N). However, in the mining process, the binary number will add 1 from 0 to 2^{M-1}, and the corresponding binary bits are judged in turn. Theoretically, the time complexity of this process is $O(2^M)$. However, the algorithm adopts a pruning optimization strategy, for which the time complexity of this process will be much lower than $O(2^M)$. The relevant experiments set up later in this paper have effectively evaluated the above analysis.

Algorithm Optimization Strategy

In practical applications, the dataset may consist of many items. As a result, the number of item combinations is large, and the corresponding calculation amount is also large. For example, in the final experiment of this paper, the transaction in the soybean promoter dataset is composed of 469 items. The corresponding number of item combinations is 2^{469} - 1, which is very large. Therefore, two strategies are used to optimize the algorithm in this paper.

Pruning Optimization Strategy

The algorithm proposed in this paper will use the addition of 1 operation in the process of mining frequent itemsets. However, if all items increase step by step from all 0 to all 1, it will lead to a large amount of calculation, so a pruning operation is needed. The pruning operation can reduce computing greatly. (Nie & Zhang, 2018).

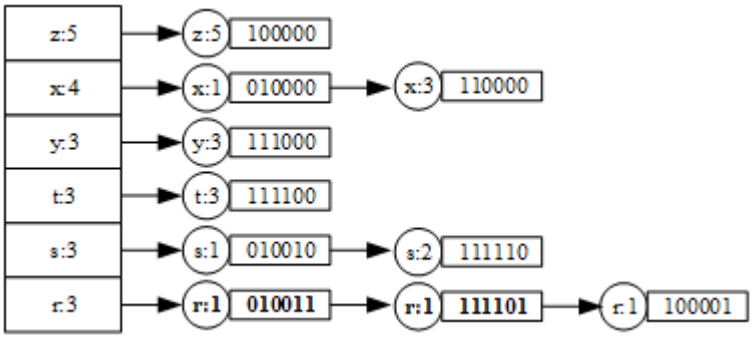
In the frequent itemset mining problem, there is a basic property, that is, all nonempty subsets of frequent itemsets must also be frequent, which is adopted by the pruning optimization algorithm. After traversing the transaction data of a certain item combination, if it is judged that the occurrence frequency of this item combination is lower than the preset support, then it is infrequent. According to the above properties, any superset containing this combination is infrequent, so it can be pruned.

For example, given a binary combination A of an itemset: 1110 1100 1100 0000. After the bitwise AND operation, if the itemset corresponding to the binary combination is infrequent, the next pruning operation will be executed. Utilizing the Formula A=A+A&(-A), the pruning operation from 1110 1100 1100 0000 to 1110 1101 0000 0000 is achieved. That is, after calculating that the itemset corresponding to the current binary combination 1110 1100 1100 0000 is infrequent, the next step is to directly judge 1110 1101 0000 0000. With the pruning strategy, many infrequent itemsets will not need to be calculated, which saves the execution time of the algorithm.

Grouping Optimization Strategy

In the process of linear table construction and frequent itemset mining, a grouping optimization strategy is introduced. The specific operation of this optimization strategy is to sort the items according to the support and then insert transaction items to build a linear table. The first item is the item with the highest support. After all the transactions including item with the highest support are inserted into the linear table, the second item with the second highest support is handled. That is, all the transactions including item with the second highest support are inserted into the linear table. Finally, the remaining transaction items are inserted. The resulting linear table will be relatively compact, and it will also be mined according to this grouping order during mining, which will improve the speed of mining frequent itemsets. The smaller the support is, the more obvious the effect of grouping optimization. After adding the grouping, the mining logic structure in Figure 4 will be updated as shown in Figure 5.

Figure 5. Group mining logical structure



After adding two optimization strategies, the pseudocode of the whole algorithm is shown in Algorithm 1.

```
Algorithm 1: pseudocode of the frequent itemset mining process
Input: complete linear table
Output: Frequent itemset combinations that match the support
Begin:
1) WHILE d[judgeI]£maxValue DO
2)  FOR j1¬0 TO judgeI
3)   FOR temp_1¬0 TO Data_Length
4)    IF (d[j1]&Array_Pow[temp_1]) THEN
5)     F[j]=usefulValue[i];
6)    END IF
7)   END FOR
8)  END FOR
9)  IF (The current itemset F is in a group) THEN
10)     update the value of last_position according to the header
table;
11)     WHILE last_position[1]0 DO
12)      IF (not equal TO itself after bitwise AND operation) THEN
13)       break
14)      END IF
15)      IF (j1<0) THEN
16)       update fre value;
17)      END IF
18)      update the value of last_position according to the Array_
point;
19)     END WHILE
20)    END IF
21)    IF (fre[3]SUPPORT) THEN
22)     obtain frequent itemset;
23)    ELSE THEN
24)     pruning strategy;
25)    END IF
26)   END WHILE
27)   RETURN frequent itemset
```

Note: maxValue is the maximum value of recording in this highest node. F is an array that stores the filtered and sorted transaction data. fre is the frequency of the item combination. Data_Length is the length of each node, and its value is 32 bits. judgeI is the largest node index of each data, and judgeI= (frequent 1-item number)/32.

EXPERIMENT

Experimental Data

The test input data used in this experiment are soybean promoter data (Zhiyong et al., 2015), which is a sparse dataset. Frequent itemsets are mined from these promoter data. Promoter analysis (Shao et al., 2019) plays an important role in the construction of gene engineering vectors and the expression of target proteins (Jing et al., 2014). Some potential connections among different regulatory elements can be identified, which can reveal secrets in genetic data. In addition, three different public datasets (URL: http://fimi.ua.ac.be) are used to validate the experiments. It lists the parameter characteristics of these datasets in Table 8, where Num_trans is the number of transactions, Distinct_items is the number of items, and Avg.length represents the average length of the transaction. Soybean is the sparse dataset, and the others are dense datasets.

Table 8. Datasets parameters

Dataset	Num_trans	Distinct_items	Size	Avg.length	Characteristic	Type
soybean	54174	469 (0-468)	6943 KB	34	sparse	Real
accidents	56698	468 (1-468)	5779 KB	33	dense	Real
chess	3196	75 (1-75)	335 KB	37	dense	UCI
mushroom	8124	119 (1-119)	558 KB	23	dense	UCI

Mining the frequent set involves finding combinations of items that meet the minimum support number min_sup (such as min_sup = 5000) and recording the support number (actual occurrence times) of relevant item combinations. For instance, in the soybean promoter dataset, it needs to select all combined elements that have occurred that are not less than min_sup in the soybean promoter dataset and actual times of occurrence. The output model is similar to 340 144 (9840).

Experimental Results and Analysis

The programs in this section run on the Linux operating system, and the gcc compiler is adopted. The chip model used in the experiments of this paper is an Intel(R) Xeon(R) CPU E5-2670 v2, and its frequency is 2.50GHz.

The frequent itemset mining algorithm based on a linear table is an accurate mining algorithm. That is, the mining accuracy is 100%. The selected comparison algorithms, the bit combination algorithm and the FP-growth algorithm, are also accurate mining algorithms. Therefore, this paper evaluates the algorithm only in terms of its running time and memory usage.

Runtime Evaluation

The running time of the algorithm in this paper is the time of constructing the linear table and mining frequent itemsets. To evaluate the time efficiency of constructing the linear table and mining, the experimental results performed on the test dataset (soybean promoter) are listed in Table 9.

Table 9. Time evaluation for constructing linear table and mining

Support	Constructing linear table (s)	Mining (s)	Total time (s)
25000	0.195	0.000	0.195
24000	0.218	0.003	0.221
23000	0.213	0.038	0.269
22000	0.259	9.893	10.152
21000	0.274	22.761	23.035
20000	0.304	523.085	523.389

It can be seen from Table 9 that as the support reduces, the time to construct a linear table is much less than the time to mine frequent itemsets. In practical applications, to better study the relationship characteristics among items in the dataset, the support is generally lower, so the time of constructing the linear table has little effect on the total time. Therefore, for simplicity, this paper no longer sets up experiments to evaluate the efficiency of constructing a linear table and mining separately. Instead, the total time of constructing and mining is adopted to evaluate the algorithm.

The frequent itemset mining algorithm based on the linear table proposed in this paper is optimized by pruning and grouping strategies, and the efficiency is significantly improved. To verify the effectiveness of the pruning strategy and the grouping strategy, the following experiments were carried out on the soybean promoter dataset. Table 10 lists the experimental comparison results before and after the pruning and grouping optimization.

Table 10 shows that the smaller the support is, the more obvious the superiority of the pruning optimization strategy. This is because the smaller the support is, the more frequent 1 itemsets that can be obtained by preprocessing, and the corresponding length of the binary combination bits will be longer. Without the pruning strategy, the binary bits will add 1 from all 0 to all 1, which will result in a very large number of mining calculations and waste considerable time. Further analysis shows that under the premise of not using pruning optimization, the smaller the support setting is, the higher the efficiency of using grouping optimization; however, the efficiency of grouping optimization is not reflected after pruning optimization is used. This is because the current support setting is relatively large, the length of the linear table constructed is relatively short, and the advantages of using group optimization cannot be reflected. For this reason, this article slowly lowered the support and continued the following experiments. The experimental results are shown in Table 11.

Table 10. Experimental comparison before and after pruning and grouping optimization

Support	No pruning and grouping (s)	Grouping (s)	Pruning (s)	Pruning and grouping (s)
25000	0.195	0.199	0.195	0.197
22500	0.380	0.344	0.233	0.235
20000	523.389	355.928	0.339	0.339

It can be seen from Table 11 that in the case of pruning optimization, continuing to use the grouping optimization strategy can further improve the algorithm mining speed because the smaller the support setting is, the more compact the linear table built by grouping, so during the mining process, the cost of scanning the linear table is reduced. Through the above experiments, it can be proven that adding pruning

Table 11. Experimental comparison before and after grouping optimization

Support	Pruning (s)	Pruning and grouping (s)
10000	1.336	1.115
7500	2.564	2.351
5000	5.998	5.723
2500	36.305	33.336

and grouping optimization strategies can indeed improve the efficiency of the algorithm, especially after adding the pruning optimization strategy, and the efficiency of the algorithm has been greatly improved.

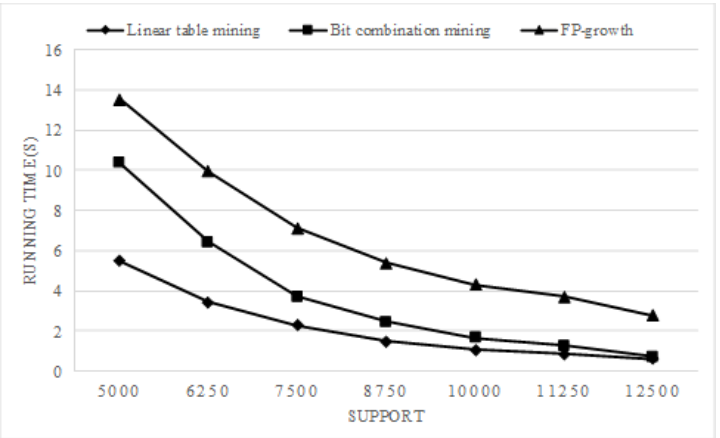
The idea proposed in the frequent itemset mining algorithm based on a linear table can avoid the recursive process in the mining process. However, it actually does what the recursion does. Compared with other mining methods, this method can perform effective parallelism, which can improve its mining speed. This mining idea also has its disadvantages. When a new candidate item is generated, the original dataset needs to be scanned once. Although optimization strategies such as "pruning" will be used during the mining process, due to too many scans, the mining speed will eventually be adversely affected. The algorithm proposed in this paper makes more use of shared information and does not need to scan the original dataset frequently during the mining process.

In the FP-growth algorithm, there is a recursive operation during the mining process. When a large dataset is processed, a large-scale tree will be generated, which will increase the calculation amount and reduce the efficiency of mining frequent itemsets. The key is that when the data are very large and the memory cannot be processed at one time, FP-growth cannot be used for batch processing. The algorithm proposed in this paper avoids the recursive operation of the FP-growth algorithm in the mining process, which is conducive to batch data processing for large datasets.

This paper compares the running time of the proposed algorithm with the bit-combination algorithm and the FP-growth algorithm on the four datasets in Table 8. Among them, the comparison of the running time of soybean promoters (sparse datasets) on the real dataset is shown in Figure 6, where the horizontal axis represents the support degree and the vertical axis represents the running time.

Figure 6 shows that the mining algorithm based on the linear table proposed in this paper is faster than the bit combination mining algorithm and the FP-growth algorithm. Especially when the support setting is relatively low, the mining speed can reach a more satisfactory level.

Figure 6. Comparison of running time on soybean



Next, the experiment is carried out on the dense dataset. Figure 7 shows the comparison of running time on the real dataset accidents.

Figure 7 shows that the frequent itemset mining algorithm based on the linear table proposed in this paper has an average improvement of 70% compared with the FP-growth algorithm and an improvement of nearly dozens of times for the bit-combination algorithm. It clearly shows the superiority of this algorithm.

Figure 8 shows the running time comparison on the chess dataset (Figure 8a) and mushroom dataset (Figure 8b). They all belong to the UCI datasets.

Figure 8 shows that the running efficiency of the frequent itemset mining algorithm based on a linear table is no less than that of the FP-growth algorithm, and it has an order of magnitude improvement over the bit combination algorithm, especially when the support is low.

Memory Evaluation

This paper further uses the dataset in Table 8 to evaluate the memory usage of each algorithm, and the experimental results are shown in Figures 9–12.

It can be seen from Figure 9 to Figure 12 that the memory usage of the bit combination mining algorithm is the least, the linear table mining algorithm proposed in this paper is second, and the

Figure 7. Comparison of running time on accidents

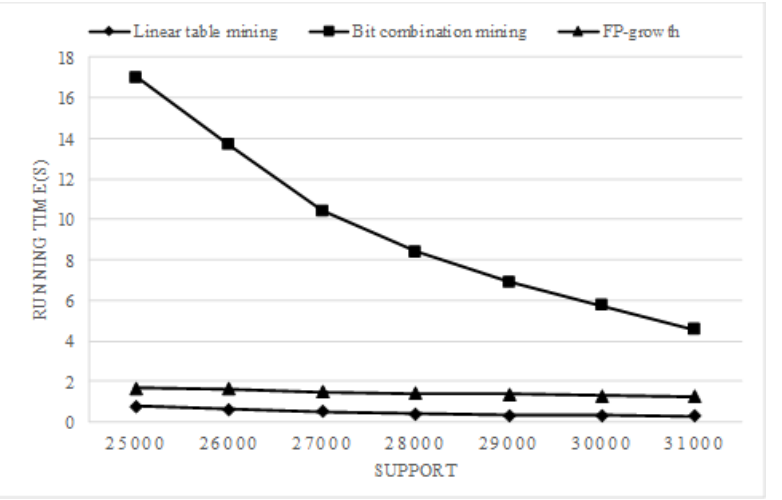


Figure 8. Comparison of running time on chess and mushroom

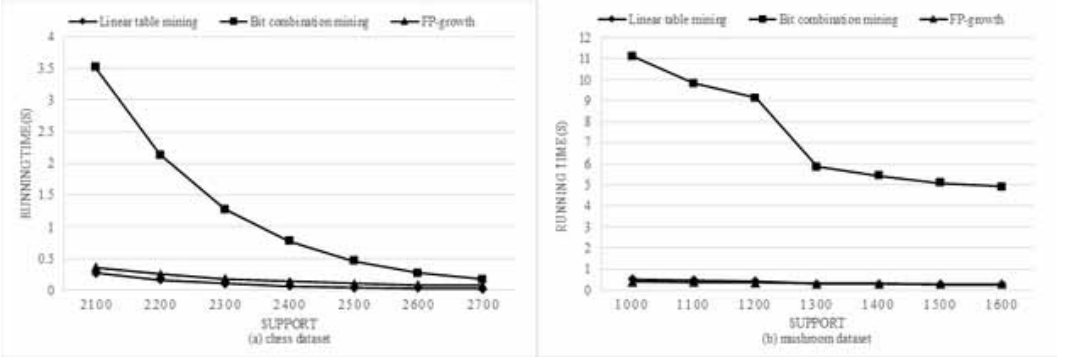


Figure 9. Memory test on soybean promoter

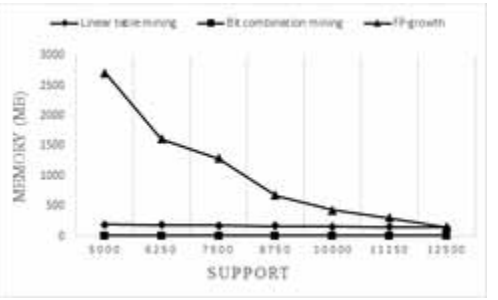


Figure 10. Memory test on chess

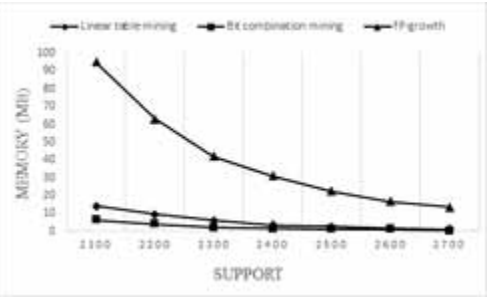


Figure 11. Memory test on accidents

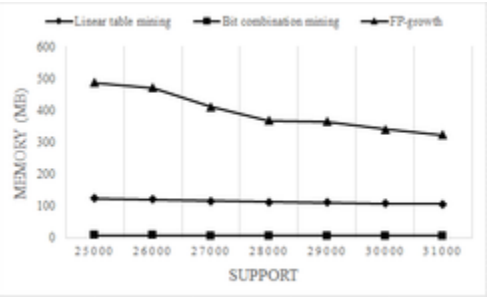
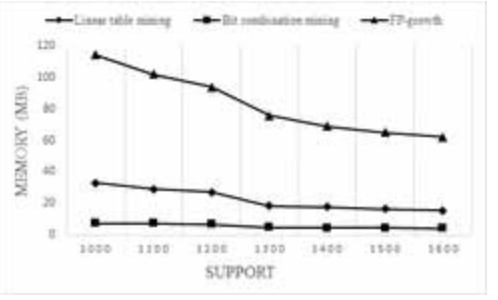


Figure 12. Memory test on mushroom



FP-growth algorithm is the most used. Although the FP-growth algorithm only scans the dataset twice, the recursive tree needs to be built continuously during the mining process. This leads to considerable memory consumption, which is why the algorithm in this paper is better than the FP-growth algorithm. On various datasets, the linear table mining algorithm uses slightly more memory than the bit combination algorithm of the previously proposed method. The linear table algorithm has improved performance on the chess dataset in Figure 10. The linear table method directly converts the original dataset into effective information in the preprocessing stage and projects it into the linear table, which reduces the number of scans in the original dataset and is beneficial for improving the mining efficiency. In contrast, bit combination algorithm requires repeated scanning of the dataset, which wastes considerable time. In a word, the bit combination mining algorithm utilizes less memory space than the linear table mining algorithm. However, further analysis from Figure 6 to Figure 8 shows that the running speed of the linear table mining algorithm has been improved by an order of magnitude at the expense of a small amount of space. In the technologically advanced world, it is significant to swiftly extract useful knowledge from massive data.

DISCUSSION

Based on the bit combination algorithm, the frequent itemset mining algorithm based on a linear table is proposed in this paper. The algorithm constructs a linear table by scanning the original dataset twice and groups it, representing the original transaction in binary form. Then cooperate with pruning technology and perform mining operations in specific groups. The linear table method eliminates the drawback of repeatedly scanning the original dataset in the bit combination algorithm and improves the performance of the algorithm.

In the process of constructing the linear table for the algorithm proposed in this paper, each newly generated item needs to store $\left(d_0, d_1, d_2, d_3, d_4, d_5, d_6\right)$ a total of seven pieces of information. However, when mining frequent itemsets, delete the d_1 child node information, d_2 brother node information, and d_3 parent node information in $\left(d_0, d_1, d_2, d_3, d_4, d_5, d_6\right)$. Because d_1, d_2 and d_3 make the linear table generation speed of the transaction data insertion process faster, and the mining process is bottom-up, deleting the node information of d_1, d_2 and d_3 will not affect the mining process. In contrast, it can reduce the space occupied by the linear table.

When dealing with large datasets, because the idea proposed by this algorithm is based on a linear table, if the original dataset is too large and the memory is insufficient, the linear table can be split. Put a part on the disk, and put only a part of the data to be processed in memory for batch processing. It can solve the problem of relatively insufficient memory for large-scale datasets.

The experimental results on the accidents, chess, and mushroom datasets confirm that the frequent itemset mining algorithm based on a linear table is the best among the three algorithms in terms of running time efficiency. Compared with the classic FP-growth algorithm, the mining efficiency of this algorithm is increased by more than 50% on average, and it is dozens of times faster than the bit combination algorithm. In terms of memory consumption, it can be seen that the frequent itemset mining algorithm based on a linear table is better than the FP-growth algorithm, but there are still some gaps with the bit combination algorithm. However, considering the runtime performance, the algorithm sacrifices some memory spaces, then gains an order of magnitude improvement in running speed, so the exchange is worthwhile.

The algorithm is effectively applied to the soybean promoter dataset in this paper, the database is a series of gene expression sequences. The application can better extract and analyze genetic data so that more valuable biological information can be obtained. This will have a good effect on increasing the yield of soybean crops and improving the quality of the species.

Through the above analysis, it can be seen that the frequent itemset mining algorithm based on a linear table is of great theoretical and practical significance for quickly discovering valuable knowledge from massive data. And the algorithm can be further extended to mining frequent closed itemsets, maximum frequent itemsets, and fault-tolerant frequent itemsets. In addition, the algorithm proposed in this paper solves the disadvantage that the recursive operation of the FP-growth algorithm in the process of mining cannot be well parallelized. It is conducive to subsequent algorithm parallelization. Adding a parallel operation can further improve the mining speed of the algorithm. It is one of the focuses of our later work.

CONCLUSION

This paper mainly studies the problem of frequent itemset mining, and a frequent itemset mining algorithm based on a linear table is proposed. The algorithm first performs preprocessing operations, constructs a linear table based on the processed dataset, and stores the information of the dataset in the linear table, which can make full use of shared information. In the subsequent frequent itemset mining, multiple scanning of the original dataset is avoided, and the cost of scanning the dataset is saved. At the same time, the linear table structure is easier to split than the tree structure, which is convenient for batch mining big data. Subsequently, this paper uses pruning and grouping strategies to optimize the construction of a linear table and frequent itemset mining based on a linear table to improve the efficiency of the algorithm, and the effectiveness of the pruning and grouping strategies is verified through experiments. Finally, further experiments show that the algorithm has different performances for different datasets. (1) Compared with the bit combination algorithm, the mining speed of this algorithm improves by 45% on average in sparse datasets and dozens of times in dense datasets. (2) Compared with the classic FP-growth algorithm, the mining speed of this algorithm is 2-3 times higher in sparse datasets and 50%-70% higher in dense datasets. Even in the worst case, the mining speed of this algorithm can be the same as that of the FP-growth algorithm. This algorithm also avoids recursive operations in the mining process. It is more conducive to the parallelization of the algorithm. All of these highlights the advantages of the algorithm in this paper.

ACKNOWLEDGMENT

Conflict of Interest

The authors of this publication declare there is no conflict of interest.

Funding Agency

This research was supported by the Graduate Innovation Research Project of Heilongjiang University [grant number YJSCX2022-233HLJU].

REFERENCES

Al-Bana, M. R., Farhan, M. S., & Othman, N. A. (2022). An Efficient Spark-Based Hybrid Frequent Itemset Mining Algorithm for Big Data. *Data*, *7*(1), 11. doi:10.3390/data7010011

Aryabarzan, N., & Minaei-Bidgoli, B. (2021). NEclatClosed: A vertical algorithm for mining frequent closed itemsets. *Expert Systems with Applications*, *174*, 114738. doi:10.1016/j.eswa.2021.114738

Aryabarzan, N., Minaei-Bidgoli, B., & Teshnehlab, M. (2018). negFIN: An efficient algorithm for fast mining frequent itemsets. *Expert Systems with Applications*, *105*, 129–143. doi:10.1016/j.eswa.2018.03.041

Bui, H., Vo, B., Nguyen-Hoang, T. A., & Yun, U. (2021). Mining frequent weighted closed itemsets using the WN-list structure and an early pruning strategy. *Applied Intelligence*, *51*(3), 1439–1459. doi:10.1007/s10489-020-01899-7

Deng, Z., Wang, Z., & Jiang, J. (2012). A new algorithm for fast mining frequent itemsets using N-lists. *Science China. Information Sciences*, *55*(9), 2008–2030. doi:10.1007/s11432-012-4638-z

Deng, Z. H. (2016). DiffNodesets: An efficient structure for fast mining frequent itemsets. *Applied Soft Computing*, *41*, 214–223. doi:10.1016/j.asoc.2016.01.010

Deng, Z. H., & Lv, S. L. (2014). Fast mining frequent itemsets using Nodesets. *Expert Systems with Applications*, *41*(10), 4505–4512. doi:10.1016/j.eswa.2014.01.025

Deng, Z. H., & Lv, S. L. (2015). PrePost+: An efficient N-lists-based algorithm for mining frequent itemsets via Children–Parent Equivalence pruning. *Expert Systems with Applications*, *42*(13), 5424–5432. doi:10.1016/j.eswa.2015.03.004

Djenouri, Y., Comuzzi, M., & Djenouri, D. (2017). SS-FIM: single scan for frequent itemsets mining in transactional databases. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining,* (pp. 644-654). doi:10.1007/978-3-319-57529-2_50

Dobra, A., & Gehrke, J. (2002). SECRET: A scalable linear regression tree algorithm. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining,* (pp. 481-487). ACM. doi:10.1145/775047.775117

Fiori, A., Grand, A., Bruno, G., Brundu, F. G., Schioppa, D., & Bertotti, A. (2014). Information extraction from microarray data: A survey of data mining techniques. [JDM]. *Journal of Database Management*, *25*(1), 29–58. doi:10.4018/jdm.2014010102

Gatuha, G., & Jiang, T. (2017). Smart frequent itemsets mining algorithm based on FP-tree and DIFFset data structures. *Turkish Journal of Electrical Engineering and Computer Sciences*, *25*(3), 2096–2107. doi:10.3906/elk-1602-113

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. *SIGMOD Record*, *29*(2), 1–12. doi:10.1145/335191.335372

Hidayanto, B. C., Muhammad, R. F., Kusumawardani, R. P., & Syafaat, A. (2017). Network intrusion detection systems analysis using frequent item set mining algorithm FP-max and apriori. *Procedia Computer Science*, *124*, 751–758. doi:10.1016/j.procs.2017.12.214

Jia-Li, X. U., Yang, H. J., Zhao, M. J., & Fan, Y. (2013). Algorithm based on bit operation for mining frequent closed itemsets. *Jisuanji Yingyong Yanjiu*, *30*(11), 3280–3279.

Jing, W., Bing, L., Liu, C., Zhen, Z., & Zhang, J. (2014). Advances of the studies on structure and function of promoter. *Shengwu Jishu Tongbao*, *46*(8), 40–45.

Lessanibahri, S., Gastaldi, L., & Fernández, C. G. (2020). A novel pruning algorithm for mining long and maximum length frequent itemsets. *Expert Systems with Applications*, *142*, 113004. doi:10.1016/j.eswa.2019.113004

Li, Z., Chen, F., Wu, J., Liu, Z., & Liu, W. (2021). Efficient weighted probabilistic frequent itemset mining in uncertain databases. *Expert Systems: International Journal of Knowledge Engineering and Neural Networks*, *38*(5), e12551. doi:10.1111/exsy.12551

Liao, S. H., Chu, P. H., & Hsiao, P. Y. (2012). Data mining techniques and applications – a decade review from 2000 to 2011. *Expert Systems with Applications*, *39*(12), 11303–11311. doi:10.1016/j.eswa.2012.02.063

Liu, J., Ye, Z., Yang, X., Wang, X., Shen, L., & Jiang, X. (2022). Efficient strategies for incremental mining of frequent closed itemsets over data streams. *Expert Systems with Applications*, *191*, 116220. doi:10.1016/j.eswa.2021.116220

Lu, J., Zhao, R., & Zhou, K. (2019). Frequent Item Set Mining Algorithm Based on Bit Combination. *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA),* (pp. 72-76). IEEE.

Nafie Ali, F. M., & Mohamed Hamed, A. A. (2018). Usage Apriori and clustering algorithms in WEKA tools to mining dataset of traffic accidents. *Journal of Information and Telecommunication*, *2*(3), 231–245. doi:10.1080/24751839.2018.1448205

Nie, Y., & Zhang, T. (2018). Improved pruning algorithm for gaussian mixture probability hypothesis density filter. *Journal of Systems Engineering and Electronics*, *29*(2), 229–235. doi:10.21629/JSEE.2018.02.02

Paladhi, S., Chatterjee, S., Goto, T., & Sen, S. (2019). AFARTICA: A Frequent Item-Set Mining Method Using Artificial Cell Division Algorithm. [JDM]. *Journal of Database Management*, *30*(3), 71–93. doi:10.4018/JDM.2019070104

Qu, J. F., Hang, B., Wu, Z., Wu, Z., Gu, Q., & Tang, B. (2020). Efficient mining of frequent itemsets using only one dynamic prefix tree. *IEEE Access : Practical Innovations, Open Solutions*, *8*, 183722–183735. doi:10.1109/ACCESS.2020.3029302

Shao, Y., Yang, M., Bao, G., Sun, Y., Yang, Q., & Li, W. (2019). Cloning and functional analysis of soybean gmwri1a gene promoter. *Chinese Journal of Oil Crops*, *41*(4), 517–523.

Vo, B., Hong, T. P., & Le, B. (2012). DBV-Miner: A Dynamic Bit-Vector approach for fast mining frequent closed itemsets. *Expert Systems with Applications*, *39*(8), 7196–7206. doi:10.1016/j.eswa.2012.01.062

Wang, Y. B. (2014). Study on data mining techniques and algorithms of association rules data mining. *Applied Mechanics and Materials*, *543*, 2040–2044. doi:10.4028/www.scientific.net/AMM.543-547.2040

Witten, I. H., & Frank, E. (2002). Data mining: Practical machine learning tools and techniques with Java implementations. *SIGMOD Record*, *31*(1), 76–77. doi:10.1145/507338.507355

Wu, H., Lu, Z., Pan, L., Xu, R., & Jiang, W. (2009). An improved apriori-based algorithm for association rules mining. *In 2009 IEEE sixth international conference on fuzzy systems and knowledge discovery,* (*vol. 2*, pp. 51-55).

Xun, Y., Cui, X., Zhang, J., & Yin, Q. (2021). Incremental frequent itemsets mining based on frequent pattern tree and multi-scale. *Expert Systems with Applications*, *163*, 113805. doi:10.1016/j.eswa.2020.113805

Yuan, X. (2017). An Improved Apriori Algorithm for Mining Association Rules. *AIP Conference Proceedings*, *1820*, 080005. doi:10.1063/1.4977361

Zahra, H. I. A., El-Sappagh, S., & El Shishtawy, T. A. (2020). A Proposed Frequent Itemset Discovery Algorithm Based on Item Weights and Uncertainty. [IJSKD]. *International Journal of Sociotechnology and Knowledge Development*, *12*(1), 98–118. doi:10.4018/IJSKD.2020010106

Zhiyong, N. I., Yuehua, Y. U., Liu, C., Ren, Y., Chen, Q., & Yanying, Q. U. (2015). Cloning of soybean promoter and construction of plant expression vector. *Journal of North China Agriculture*, *30*(1), 19–23.

Wenhe Xu graduated from Jiangxi University of Science and Technology. He is currently studying at the College of Computer Science and Technology, Heilongjiang University, majoring in data mining.

Zekun Wang She is currently studying at the College of Physics, Changchun University of Science and Technology. Her main research interests include data mining and Optoelectronic information engineering.

Jun Lu received a Ph.D. degree in computer application and technology from Harbin Engineering University in 2010. She is currently a professor at Heilongjiang University. Her main research interests include data mining, machine learning, and data engineering.

Kailong Zhou graduated from Heilongjiang University. His main research interest is data mining.

Zhicong Guo graduated from Heilongjiang University. His main research interest is data mining.