DOI: 10.21917/ijsc.2025.0510

EFFICIENT FREQUENT ITEMSET DISCOVERY THROUGH HIERARCHICAL HUFFMAN ENCODING

Dai Xin and Hao Xue
Faculty of Computing, Universiti Teknologi Malaysia, Malaysia

Abstract

Frequent itemsets mining holds a crucial position in the field of data mining; however, traditional algorithms like Apriori and FP-Growth often encounter efficiency and memory consumption issues when handling large-scale datasets, which not only makes them difficult to cope with dynamic dataset changes in some situations but also limits their widespread use in practical applications. Therefore, a novel DTFIMA (Dynamic Tiered Frequent Itemset Mining Algorithm,) algorithm is proposed in this paper to address optimization problems related to the storage and searching process of frequent itemsets by introducing dynamic weight Huffman coding and combining it with logarithmic frequency stratification. In ADFIM, all itemsets are divided into three frequency levels: high, medium, and low, and independent Huffman trees are created for each level, thus achieving higher efficiency in frequent itemsets encoding and search. Meanwhile, ADFIM improves the accuracy of frequent itemset mining and enhances the algorithm's stability and reliability in handling large-scale data by dynamically updating weights and timely cleaning low-frequency items. Experimental results show that compared to the traditional FP-Growth algorithm, ADFIM demonstrates higher efficiency in handling large-scale transaction databases, especially in dealing with dynamic data streams, significantly reducing computation time while ensuring the accuracy and consistency of frequent itemset discovery.

Keywords:

Frequent Itemsets Mining, Huffman Coding, Logarithmic Frequency Stratification, Dynamic Data Streams

1. INTRODUCTION

. Frequent itemset mining plays an indispensable role in many fields, whether in market basket analysis, intrusion detection or recommendation systems, and existing frequent itemset mining algorithms, such as Apriori and FP-Growth, although it is widely used and has achieved certain success, each also has obvious advantages and disadvantages, which in some cases will significantly affect the performance and applicability of the algorithm. Specifically, the Apriori algorithm due to its needs the large computational overhead and multiple scans of the database make it unscalable when processing large-scale data sets, and even completely unfeasible in some cases [1].

On the other hand, the FP-Growth algorithm reduces database scans by building FP trees and it performs well when processing certain types of data. However, when faced with sparse data sets, the FP tree it constructs may be too large and complex, resulting in a significant decrease in algorithm efficiency [2]. This not only increases memory usage, but also it may cause the algorithm to run too long, causing FP-Growth to have various limitations when facing sparse data [3].

In response to the above problems, this paper proposes a new frequent itemset mining algorithm, which combines dynamic weighted Huffman coding and hierarchical dynamic partitioning mechanism (Tiered Dynamic Weighted Frequent Itemset Mining Algorithm, TDWFIMA) to optimize the item set. Storage and search processes. TDWFIMA divides all items into three levels: high frequency, medium frequency, and low frequency through a hierarchical coding strategy, and creates a Huffman tree for each level, thus greatly improving the efficiency of coding and searching. The advantage of this method is that it combines the efficient compression characteristics of Huffman coding and the dynamic adaptation characteristics of frequent itemset mining, which has key execution time and memory advantages compared with classic algorithms. Its contributions include:

- Introducing innovative dynamic weighted Huffman coding. By dynamically adjusting weights and hierarchical coding strategies, the coding length of high-frequency items is effectively reduced and search efficiency is optimized [4].
- Through the hierarchical dynamic partitioning mechanism, the item set search process is made more efficient and the overhead of multiple database scans is avoided [5].
- Through experiments on public data sets, it is verified that the algorithm has significantly superior performance compared with traditional methods. Experimental results show that the algorithm using TDWFIMA is superior to the traditional algorithm in terms of execution time and memory usage. Experimental results show that the TDWFIMA algorithm can mine frequent item sets more efficiently when processing dynamic data streams, significantly reducing computational overhead and improving memory utilization. These advantages give it broad potential for practical applications.

This research paper discusses in detail the Tiered Dynamic Weighted Frequent Itemset Mining Algorithm (TDWFIMA) that combines dynamic weighted Huffman coding with hierarchical dynamic partitioning and conducts in-depth research on its application potential in the field of frequent item set mining, showing its potential to provide an efficient solution for data mining and opening up new research directions.

The rest of this paper is arranged as follows: The second part will thoroughly analyze the problems existing in the current frequent item set mining algorithm, explore its limitations and room for improvement from multiple perspectives, and the third part will introduce the basic principles, design ideas and implementation process of the TDWFIMA algorithm in detail, showing its unique algorithmic advantages and innovations. The fourth part focuses on performance evaluation. Through a large amount of experimental data and comparative analysis, it proves the superior performance of TDWFIMA in processing large-scale data. The fifth part summarizes the full text, draws conclusions, and proposes future research directions and potential application scenarios.

2. RELATED WORK

The research direction of frequent item set mining occupies an extremely important position in the field of data mining. Its core purpose is to find out those itemsets that often co-occur from a large amount of transaction data. After the market basket analysis theory proposed by Agrawal et al., this research field quickly attracted widespread attention and has been deeply studied. Researchers in many fields continue to propose various algorithms to improve further the efficiency and scalability of frequent itemset mining, based on different technical means and optimization strategies to meet the processing needs of complex data, including but not limited to Apriori algorithm, FP-Growth algorithm, Eclat algorithm, etc. In continuous development and evolution, these algorithms strive to make breakthroughs and improvements in computing speed, memory consumption, and mining accuracy, so that frequent itemset mining technology can better adapt to the diverse and rapidly changing data needs in actual application scenarios.

2.1 LIMITATIONS OF TRADITIONAL FREQUENT ITEMSET MINING ALGORITHMS

In the research field of frequent itemset mining, many classic algorithms have been widely used and achieved certain success. These algorithms mainly include Apriori algorithm, FP-Growth algorithm and Eclat algorithm [6]. However, with the continuous expansion of data scale and the real-time demand of data flow, these traditional algorithms have many limitations when processing dynamic data and large-scale data sets [7]. For example, the Apriori algorithm needs to scan the entire database multiple times to generate frequent item sets. This process not only generates a lot of I/O overhead but also causes huge computational overhead because the algorithm needs to generate a large number of candidate item sets in each iteration [8]. In contrast, although the FP-Growth algorithm compresses data by constructing a frequent pattern tree (FP tree), which significantly reduces the number of database scans and the number of candidate item sets generated when processing large-scale data sets, the construction and storage of the FP tree still require a lot of memory resources [9]. In addition, FP-Growth needs to recursively mine the FP tree. For tree structures with deeper levels, the recursive operation is complicated and easily leads to stack overflow. To conquer these restrictions related to classic algorithms, researchers have proposed many improved algorithms, such as those based on compression technology, distributed computing, and sliding window techniques [10]. Although these improved algorithms have greatly enhanced the efficiency of frequent itemset mining to some extent, they still cannot update the frequent itemset in real-time when processing dynamic data stream streams and leads to inaccurate results [11].

2.2 HIERARCHICAL FREQUENT ITEMSET MINING

In the field of frequent itemset mining (FIM), traditional algorithms such as Apriori and FP-Growth are inefficient when processing large-scale data, and their memory consumption is extremely large [1]. To deal with these problems, the Hierarchical Frequent Itemset Mining technology came into being. It hierarchizes frequent itemsets according to specific criteria, thereby improving the efficiency and effect of frequent itemsets mining to a certain extent [12]. The core idea of this method is to optimize the mining process by organizing frequent itemsets hierarchically, trying to solve the limitations of traditional algorithms when processing large-scale data. In the weight-based layering method, Stumme et al. proposed a method of assigning weights to itemsets and layering them according to the weights. Although this method can effectively reduce the interference of low-frequency itemsets, it is difficult to maintain high efficiency when facing dynamically changing data sets [13]. The layering method based on support is another common method. For example, the H-mine algorithm proposed divides frequent itemsets into different levels according to support by constructing a hierarchical index structure, thereby improving mining efficiency [14]. However, this method performs well when processing static data sets. When facing dynamic data streams, changes in support require frequent updates of the hierarchical structure, which increases the computational complexity. In addition, to adapt to the needs of dynamic data sets, Jiang and Gruenwald proposed a dynamic hierarchical algorithm that divides frequent itemsets into high-frequency, medium-frequency, and low-frequency levels by real-time monitoring and adjustment of frequent itemsets[15]. This method can respond to data changes promptly, but in high-frequency data update scenarios, the computational overhead is large. In our method, by combining dynamic weighted Huffman coding and hierarchical dynamic segmentation, by dynamic weighting and hierarchical processing of frequent itemsets, Not only the mining efficiency is improved, but also the data changes can be responded to in real-time, and excellent performance is achieved when facing large-scale dynamic data sets.

2.3 APPLICATION AND LIMITATIONS OF HUFFMAN CODING IN FREQUENT ITEMSET MINING

The application of Huffman coding in frequent itemset mining is a complex and multi-level problem, which not only involves the efficient compression characteristics of coding, but is also closely related to the fast retrieval and storage optimization of frequent itemsets. Specifically, Huffman coding achieves efficient compression by assigning shorter codes to high-frequency items, thereby significantly saving storage space, which is particularly evident when processing large-scale data sets [16]. However, the application of Huffman coding in dynamic data stream environments faces many challenges. Traditional Huffman coding is generated based on static data sets, while in dynamic data stream environments, the frequency of items changes over time, which requires frequent reconstruction of the Huffman tree, increasing computational overhead and complexity [17]. After each data update, the frequency of the items needs to be recalculated and the Huffman tree needs to be rebuilt, which is unrealistic in applications with high real-time requirements [18]. In addition, Huffman coding can only encode according to the frequency of the current items and cannot foresee frequency changes in future data streams. Therefore, it may not provide the optimal coding scheme in a dynamic environment. This local optimality problem further limits its application in dynamic data stream environments [19]. Huffman coding is one of the standard data compression algorithms. The basic idea is to build a binary tree based on the frequency of characters and try to push the characters with higher frequency to the top of the tree to reduce the total length of the code [20]. By assigning concise codes to high-frequency itemsets, the number of comparisons during the search process can be reduced, thereby improving the efficiency of the algorithm. Recently, researchers are exploring possibilities of incorporating Huffman coding into data mining. Wang et al. [21] advanced in the reduction of memory overhead and computational complexity, especially when it came to dealing with large data sets of transaction data, and also clearly described how to use Huffman coding to improve the efficiency of frequent itemset mining. However, it can only improve the efficiency of frequent item set mining to some degree while the data with more extensive scale and higher dimensions are processed. A hierarchical coding strategy, on the other hand, can handle this kind of problem. Conversely, hierarchical coding techniques partition data into layers according to frequency or other criteria, enabling independent coding of each layer. Thus, not only can the approach perform further data compression, but it also provides more options to be optimized by the search process.

3. TDWFIMA ALGORITHM

3.1 PRINCIPLE

In this study, we designed a frequent itemset mining algorithm (TDWFIMA) based on dynamic weighted Huffman coding and hierarchical dynamic partitioning, which aims to deal with the concept drift detection problem in large-scale data sets. In order to better understand the operation of the algorithm, we first need to understand the definition of Huffman Node, where each node contains item, frequency, weight and timestamp information, and its weight can be dynamically updated through a formula. This design allows us to flexibly adjust the weight according to the dynamic changes of frequent itemsets, ensuring that the algorithm can respond to data changes in real time.

- **Definition 1:** The weight update formula of the node weight is as follows:

$$\text{new\_weight}=\text{node.weight}\times(1+\alpha\times\delta f_{\text{req}})\times\beta^{(\text{time\_diff}/3600)} \quad (1)$$

Among them, α is the influencing factor of frequency change, β is the time attenuation factor, δfreq is the frequency change rate, and time_diff is the time difference.

Dynamic weighted Huffman coding is not only used for efficient storage of frequent item sets, but also for layering data sets by frequency. At the same time, the coding is optimized through the Huffman tree structure to ensure more efficient data compression and retrieval. On this basis, the TDWFIMA algorithm also includes a series of key steps, including indexing, window size, indexing, item frequency, encoding, node mapping, previous frequency and frequency difference. By building index and frequency dictionaries, large-scale transaction data can be effectively organized and managed. In the process of generating the Huffman tree, a priority queue is first built based on the frequency of the item, and then the tree structure is gradually established by merging the nodes with the lowest frequency, and finally a one-fork tree structure of the root node is formed. This process is further optimized by dynamic segmentation, in which the items are divided into three levels of high, medium and low according to the logarithmic frequency, so that frequent item sets can be better managed.

- **Definition 2:** The Hierarchical Dynamic Partitioningas follows:

Compute the log-frequency of itemsets:

$$\log f_i=\log(\sigma(i)+1) \quad (2)$$

where $\sigma(i)$ represents the frequency of item i.

Compute the total log frequency:

$$Total=\sum_{i=1}^{n}\log f_i \quad (3)$$

where n is the total number of itemsets.

- **Determine the Split Point**:

High frequency layer: The cumulative sum of logarithmic frequencies is less than one third of the total

$$split_1=\frac{\sum_{i=1}^{n/3}\log f_i}{\sum_{i=1}^{n}\log f_i} \quad (4)$$

Medium frequency layer: The cumulative sum of logarithmic frequencies is between one third and two thirds.

$$split_2=\frac{\sum_{i=1}^{2n/3}\log f_i}{\sum_{i=1}^{n}\log f_i} \quad (5)$$

Low frequency layer: The cumulative sum of logarithmic frequencies is greater than two-thirds.

By dynamically updating weights and hierarchical dynamic partitioning methods combined with a sliding window mechanism, efficient management and mining of frequent itemsets are achieved.

3.2 GENERATION PROCESS

The TDFIMA (Tiered Dynamic Frequent Itemset Mining Algorithm) effectively extracts all frequent item sets with minimal support from a database by introducing dynamic weighted Huffman coding and tiered dynamic partitioning. Initially, it scans each transaction in the database, builds an inverted index, and records the occurrence positions for each item while calculating their frequencies. Subsequently, the data is organized based on frequency and segmented into high, medium, and low tiers. Each tier generates a Huffman tree that optimizes coding and searching. The algorithm utilizes dynamic weighted Huffman coding to adjust the weights based on frequency changes over time and apply time decay, ensuring the coding reflects the actual distribution of frequent items. The tiered dynamic partitioning method uses logarithmic frequency division and adaptive adjustments to capture the distribution of frequent items accurately and prioritize high-frequency items in querying and storage. While mining frequent item sets, the algorithm locates the item positions using the inverted index, compresses data with Huffman coding to minimize memory usage, and dynamically adjusts item weights. Candidate item sets are expanded step-by-step, and their support is calculated using a depth-first search (DFS) strategy. If the support of the candidate item set is not less than the pre-set minimum support threshold, the item set is considered frequent. The structural optimization of TDFIMA allows for effective frequent itemset mining when processing large-scale data, as shown in Fig.1.

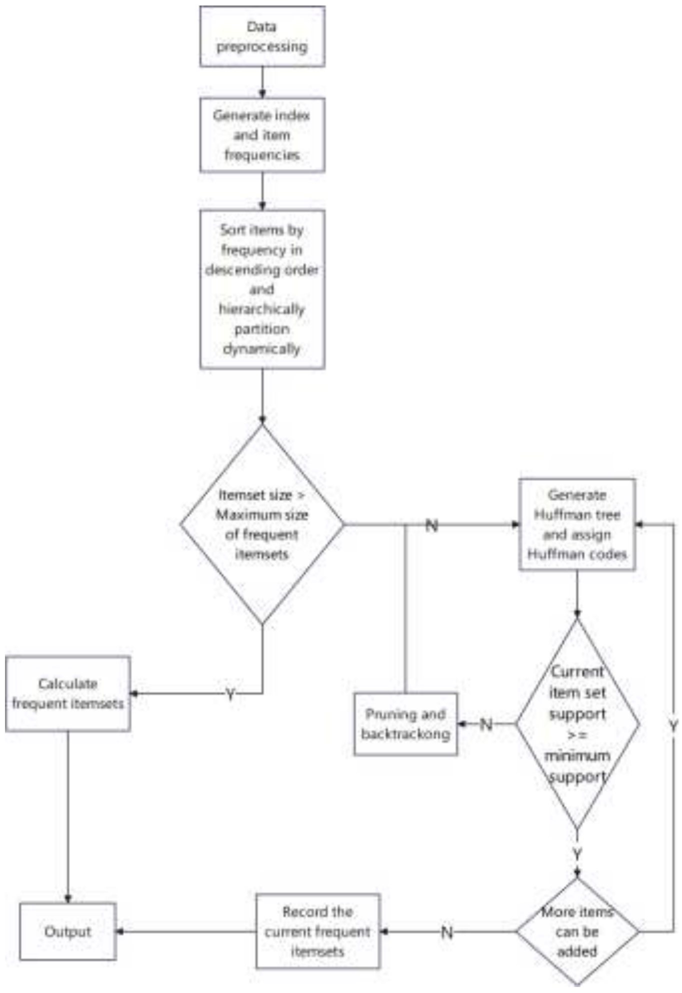

Data preprocessing
Generate index and item frequencies
Sort items by frequency in descending order and hierarchically partition dynamically
Itemset size > Maximum size of frequent itemsets
N
Generate Huffman tree and assign Huffman codes
Y
Calculate frequent itemsets
Pruning and backtrackong
N
Current item set support >= minimum support
Y
Y
Output
Record the current frequent itemsets
N
More items can be added

Fig.1. TDFIMA Algorithm structure diagram

The TDWFIMA algorithm first loads the transaction dataset. It initializes data structures such as the inverted index and item frequency dictionary (lines 3-4). It then scans each transaction, converts it into a set to remove duplicates, and updates the inverted index and item frequency dictionary accordingly (lines 5-12). During this process, the frequency of each item is counted based on its occurrence in the transaction (line 10). After calculating the item frequency, the items are sorted in descending order based on the item frequency (line 13) and divided into high, medium, and low frequency layers using a hierarchical dynamic partitioning technique, which is calculated based on the logarithmic value of the item frequency (line 14).Next, the algorithm generates a Huffman tree and assigns Huffman codes to each layer to facilitate efficient storage and retrieval of these frequent itemsets (line 15). The frequent itemset dictionary is initialized as an empty dictionary, and the algorithm sets up a stack to facilitate depth-first search of frequent itemsets (lines 16-17). The main mining process takes place in a while loop that continues until the stack is empty (lines 18-29). In this loop, prefixes, items, and the current transaction index are popped from the stack, the prefix is extended by the additional item, and the new transaction index is calculated as the intersection of the current index and the corresponding item in the inverted index (lines 20-22). If the length of the new transaction index meets the minimum support threshold, the new prefix and its support count are added to the frequent itemset dictionary, and the items with sufficient support among the remaining items are identified (lines 23-25). Subsequently, these new prefixes, remaining items, and new transaction indexes are pushed back to the stack for further exploration (line 26). Finally, after all frequent itemsets have been discovered, the algorithm returns the dictionary containing these frequent itemsets (line 30).The pseudo code is shown in Table 1

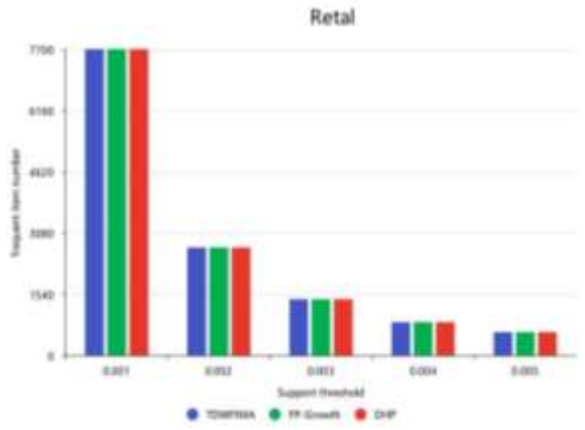
Table.1. TDWFIMA Algorithm

TDWFIMA Algorithm

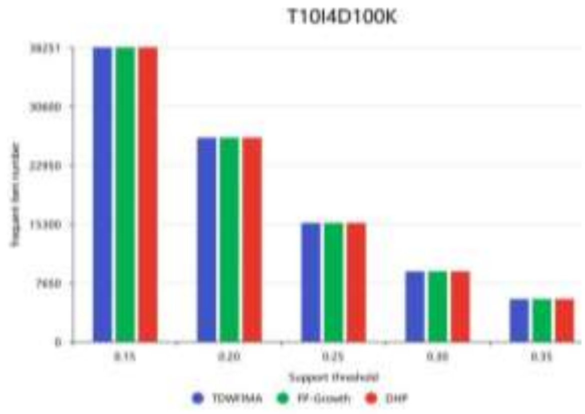
```
1: Begin
2: Load transactions from file
3: Initialize inverted_index and item_frequency dictionary
4: For each transaction Ti in transactions do
5:     Convert Ti to a set to remove duplicates
6:     For each item in Ti do
7:         If item not in inverted_index then
8:             Initialize inverted_index[item] as an empty set
9:         Add transaction index to inverted_index[item]
10:        Increment item frequency in item_frequency
11:    End For
12: End For
13: Sort items in descending order of frequency
14: Split sorted items into high, medium, and low frequency tiers
15: Generate Huffman tree and assign codes for each tier
16: Initialize frequent_itemsets as an empty dictionary
17: Initialize stack with empty prefix, sorted items, and all transaction indices
18: While stack is not empty do
19:     Pop prefix, items, and current transaction indices from stack
20:     For each item in items do
21:         Create new prefix by appending item
22:         Calculate new transaction indices as intersection of current indices and inverted_index[item]
23:         If length of new transaction indices >= min_support then
24:             Add new prefix to frequent_itemsets with its support count
25:             Find remaining items with enough support
26:             Push new prefix, remaining items, and new transaction indices onto stack
27:         End If
28:     End For
29: End While
30: Return frequent_itemsets
31: End
```

4. RESULTS AND DISCUSSION

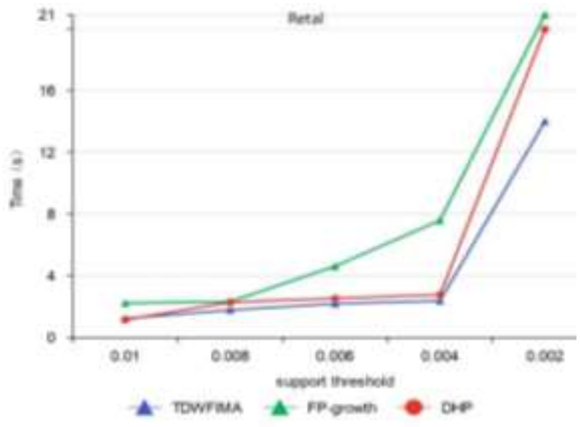
In this section, we verify the effectiveness of the proposed TDWFIMA algorithm through a detailed analysis of experimental results. All experiments were conducted on a computer with an Intel Core i5 processor (2.2 GHz), 8GB RAM, and a 64-bit Windows 10 Pro operating system, and the experiments were implemented in the Python programming language. In order to comprehensively evaluate the performance of the TDWFIMA algorithm in frequent itemset mining, we selected two public datasets, namely Retal and T10I4D100K, and compared the performance of the TDWFIMA algorithm with the classic FP-Growth and DHP (Direct Hashing and Pruning, an improved version of Apriori, using hashing technology to reduce candidate sets). The evaluation indicators mainly include the execution time of the algorithm and the performance under different support thresholds. Through these comparisons, we aim to show the advantages and potential improvement space of the TDWFIMA algorithm in dealing with frequent itemset mining tasks.



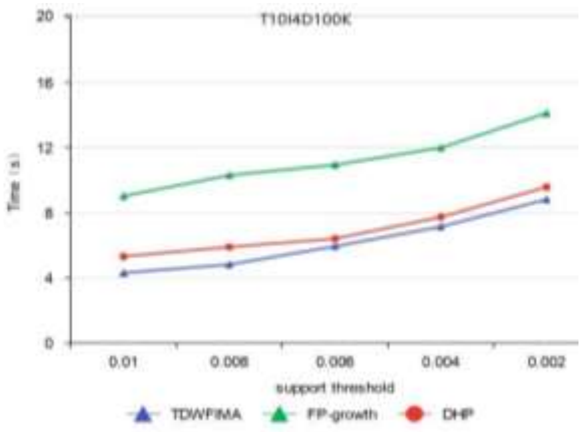
(a) The number of item sets generated by the three algorithms in the retal database



(b) The number of item sets generated by the three algorithms in the T10I4D100K database



(c) Run time comparison in the retal database



(d) Run time comparison in the T10I4D100K database

Fig.2. Experimental results diagram

From the experimental results, first, in terms of accuracy, the TDWFIMA algorithm performs well. Under different support thresholds, it can achieve the same high standards as traditional algorithms such as FP-Growth and DHP. This means that whether it is a high-frequency item or a low-frequency item, the TDWFIMA algorithm can accurately mine all frequent item sets that meet the support requirements, thereby proving its reliability and robustness in maintaining the accuracy of the results; secondly, in terms of running speed, the TDWFIMA algorithm shows significant advantages. Whether under high support thresholds or low support thresholds, the running time of TDWFIMA is significantly lower than that of traditional frequent item set mining algorithms. This feature is particularly prominent when processing large-scale data sets, indicating that it has significant improvements in optimizing computational efficiency and reducing execution time. Through the combination of dynamic weighted Huffman coding and hierarchical dynamic partitioning technology, the TDWFIMA algorithm can effectively narrow the search space, thereby significantly improving the mining speed without affecting the accuracy. This high efficiency enables TDWFIMA to better meet the needs of large-scale data analysis in practical applications.

5. CONCLUSION

This paper introduces a TDWFIMA algorithm to maximize the efficiency of frequent itemset mining with Hierarchical Dynamic Partition ingas and Hierarchical Huffman coding. Experimental results show that the execution time of the proposed TDWFIMA algorithm for T10I4D100K and the Retal dataset is impressively better in comparison with FP-Growth and DHP algorithms, which proves its effectiveness and scalability for large-scale and complex datasets. Frequent itemset mining is an important data mining technique, and the TDWFIMA algorithm provides a novel, efficient solution demonstrating potential in practical application. Future research will focus on refining this algorithm and extending its application to other data mining tasks like association rule mining and classification, with the aim of advancing high-performance data mining technology.

REFERENCES

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules", *Proceedings of International Conference on Very Large Data Bases*, pp. 487-499, 1994.

[2] J. Han, J. Pei and Y. Yin, "Mining Frequent Patterns without Candidate Generation", *Sigmod Record*, Vol. 29, No. 2, pp.1-12, 2000.

[3] M.J. Zaki, "Scalable Algorithms for Association Mining", *Transactions on Knowledge and Data Engineering*, Vol. 12, No. 3, pp. 372-390, 2000.

[4] A. Moffat, "Huffman Coding", *Computing Surveys*, Vol. 52, No. 4, pp. 1-35, 2019.

[5] S.T. Klein, S. Saadia and D. Shapira, "Forward Looking Huffman Coding", *Theory Computing Systems*, Vol. 65, pp. 593-612, 2021.

[6] S. Wang and H. Dutta, "Parable: A Parallel Random-Partition based Hierarchical Clustering Algorithm for the MapReduce Framework", *Center for Computational Learning Systems Columbia University*, pp. 1-20, 2011.

[7] H.N. Dai, R.C.W. Wong and H. Wang, "Big Data Analytics for Large-Scale Wireless Networks: Challenges and Opportunities", *Computing Surveys*, Vol. 52, No. 5, pp. 1-36, 2019.

[8] M. Bai, X. Wang and J. Xin, "An Efficient Algorithm for Distributed Density-based Outlier Detection on Big Data", *Neuro Computing*, Vol. 181, pp. 19-28, 2016.

[9] M. Chen, X. Gao and H.F. Li, "An Efficient Parallel FP-Growth Algorithm", Proceedings of International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, pp. 283-286, 2009.

[10] G. Lee, U. Yun and K.H. Ryu, "Sliding Window based Weighted Maximal Frequent Pattern Mining Over Data Streams", *Expert Systems with Applications*, Vol. 41, No. 2, pp. 694-708, 2014.

[11] H. Nam, U. Yun and E. Yoon, "Efficient Approach of Recent High Utility Stream Pattern Mining with Indexed List Structure and Pruning Strategy Considering Arrival Times of Transactions", *Information Sciences*, Vol. 529, pp. 1-27, 2020.

[12] C.J. Lee, C.C. Hsu and D.R. Chen, "A Hierarchical Document Clustering Approach with Frequent Itemsets", *Proceedings of International Conference on Cognitive Informatics and Cognitive Computing*, pp. 26-33, 2017.

[13] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier and L. Lakhal, "Computing Iceberg Concept Lattices with Titanic", *Data Engineering*, pp. 189-222, 2002.

[14] A. Borah and B. Nath, "Comparative Evaluation of Pattern Mining Techniques: An Empirical Study", *Complex and Intelligent Systems*, Vol. 7, pp. 589-619, 2021.

[15] N. Jiang and L. Gruenwald, "RHPTree-Risk Hierarchical Pattern Tree for Scalable Long-Term Frequent Pattern Mining", *Transactions on Knowledge Discovery from Data*, pp. 1-25, 2022.

[16] M.H. Dunham, Y. Xiao and L. Gruenwald, "A Survey of Association Rules", *Survey Journal*, pp. 1-65, 2000.

[17] G. Psaila and P.L. Lanzi, "Hierarchy-based Mining of Association Rules in Data Warehouses", *Proceedings of Symposium on Applied Computing*, Vol. 1, pp. 307-312, 2000.

[18] P. Kumari and M. Saini, "Anomaly Detection in Audio with Concept Drift using Dynamic Huffman Coding", *Sensors Journal*, Vol. 22, No. 17, pp. 17126-17138, 2022.

[19] P. Fang, S. Webb, Y. Chen, Y.F. Liu and P.C. Sen, "A Multiplexing Ripple Cancellation LED Driver with True Single-Stage Power Conversion and Flicker-Free Operation", *Transactions on Power Electronics*, Vol. 34, No. 10, pp. 10105-10120, 2019.

[20] Q.P. Kumari and M. Saini, "Anomaly Detection in Audio with Concept Drift using Dynamic Huffman Coding", *Sensors Journal*, Vol. 22, No. 17, pp. 17126-17138, 2022.

[21] R.M. Wang, W. Fu, X. He, S. Hao and X. Wu, "A Survey on Large-Scale Machine Learning", *Transactions on Knowledge and Data Engineering*, Vol. 34, No. 6, pp. 2574-2594, 2022.