A MULTITHREAD, NOVEL PATTERN BASED ALGORITHM FOR FINDING FREQUENT PATTERNS WITH JAGGED ARRAY AND VERTICAL DATA FORMAT

P.Sumathi
Research Scholar, PG & Research Department of Computer Science, Nehru Memorial College (Autonomous) (Affiliated to Bharathidasan University), Puthanampatti-621 007, Tiruchirappalli-Dt, Tamil Nadu, India
sumiparasu@gmail.com

Dr.S.Murugan
Associate Professor, PG & Research Department of Computer Science, Nehru Memorial College (Autonomous) (Affiliated to Bharathidasan University), Puthanampatti-621 007, Tiruchirappalli-Dt, Tamil Nadu, India
murugan_nmc@hotmail.com

Dr.V.Umadevi
Assistant Professor, PG & Research Department of Computer Science, Nehru Memorial College (Autonomous) (Affiliated to Bharathidasan University), Puthanampatti-621 007, Tiruchirappalli-Dt, Tamil Nadu, India
yazh1999@gmail.com

Abstract

Frequent pattern mining is essential for discovering hidden items from a database with more than a prescribed threshold. Knowing frequent patterns helps us to determine the relationship between the items. Many researchers narrated novel algorithms for sequential frequent itemset mining using a single thread, but still, there is a need for time, memory efficient and scalable one. Therefore, the research study proposed an approach for finding frequent patterns, namely TB-NPF-VDF (Thread Based, Novel Pattern Formations with Vertical Data Format), which uses a new way of generating candidate items to minimize the time. Also, it employs a multithread concept and runs several threads simultaneously, one for each frequent 1-itemset to generate the remaining frequent itemsets for that item. Further, it also employs a jagged array to store the frequent patterns to reduce the memory requirement. The research work has been implemented and tested using four real-time datasets. Further, it has been compared with Matrix-Apriori, VDF and NPF-VDF (without multithread), and the experimental results reveal that TB-NPF-VDF outperforms significantly in terms of runtime and storage.

***Keywords*: Frequent Patterns; Jagged Array; Multithread; Novel Pattern Formation; Vertical Data Format.**

1. Introduction

Data Mining (DM) is the fastest growing field [1], whose primary goal is to discover or extract information or patterns from large datasets. It is a multidisciplinary field comprising Computer Science and Statistics. It is an analysis step of Knowledge Discovery from Databases (KDD) [2]. Several DM techniques are available, such as Association Rule Mining (ARM), sequential pattern analysis, classification, and clustering. ARM is one of the most widely used techniques for knowledge discovery in the mining domain [3]. ARM is used in several applications such as inventory control, mobile mining, educational mining, market basket analysis, risk management, telecommunication networks and graph mining, etc. [4]. Frequent patterns are the patterns that occur frequently in a dataset whose frequency is more than that of a threshold value specified by the user. For instance, a set of items viz., pen and paper appears frequently together in a transactional dataset is a frequent itemset [1]. Mining frequent patterns is an essential sub-task of ARM [5]. It generates qualitative knowledge, which helps the decision-makers for making valuable business insights [2].

Apriori is a classical algorithm for finding frequent patterns which uses a horizontal format approach proposed by Agrawal and Srikant in 1993 [6] for Boolean association rules. The algorithm begins with generating a 1-itemset, recursively produces a frequent 2-itemset, frequent 3-itemset, and so on until all frequent itemsets are produced [4]. The main drawback of the algorithm is that it generates numerous candidate itemset,

especially for huge frequent 1-itemset and needs to scan the database many times. Many algorithms have evolved over the years to overcome these drawbacks viz., FP-growth, Direct Hashing and Pruning (DHP), Matrix-Apriori and maximal association rule mining, so on. In this line, this research work also introduces a paradigm for finding frequent patterns with a multithreaded approach.

The remaining article is organized as follows. The relevant work related to the proposed work is illustrated in Section 2. Section 3 elaborates the proposed methodology with an analogy. Section 4 discusses the results and section 5 summarizes the conclusion.

2. Related Work

The problem of mining frequent patterns is an essential task in ARM. Several studies have been carried out in this domain to improve the time to generate frequent itemsets and reduce the memory space over the years. This section presents a brief overview of them, providing a strong impetus to the proposed method.

Y. M. Guo et al. [1] have initiated a VDF algorithm for mining frequent itemsets. The new algorithm only needs a single scan of the entire database and uses AND operation for finding the frequent itemsets. Additionally, it proved that the algorithm requires less storage and also improves the mining efficiency. Subashini et al. [4] have studied ARM methods in horizontal and vertical data format approaches viz., Apriori, APRIORITID, APRIORI_RARE and APRIORIRARE_TID. They analyzed the pros and cons of each technique.

Judith Pavón et al. [7] have introduced a method called Matrix-Apriori to increase the speed of finding frequent itemsets. It first generates a Boolean matrix MFI which holds the frequent 1-itemset by traversing the transaction database. The vector STE stores the support count of the candidate itemset for each row in MFI. To accelerate the search of frequent patterns, the first row of MFI writes the indexes. It used a conditional pattern generation method for generating frequent patterns and proved that it performs better than Apriori and FP-Growth algorithms. Sumathi, P and Murugan, S [8] have designed a memory-efficient VDF approach using a jagged array and developed a memory usage model. They demonstrated that memory usage was reduced significantly when compared with multidimensional arrays.

A fast GPU-based frequent itemset mining algorithm for massive datasets called GMiner has been introduced in [9]. It has been developed to overcome the limitations of various parallelism methods viz., multi-core CPU, multiple machines and many-core GPU, particularly the workload skewness. It extracts the patterns from the enumeration tree and uses the computational power of GPU. From the experimentation, they showed that the GMiner is better than the existing ones. Authors in [10] have suggested a novel algorithm, namely Accelerating Parallel Frequent Itemset Mining on Graphics Processors with Sorting (APFMS). This parallel frequent itemset mining utilizes GPU's to accelerate the mining process. GPUs speed-up process using the OpenCL platform and proved that the APFMS outperforms the previous computation time-based methods.

A new multi-core based parallel mining algorithm for finding frequent itemsets has been presented in [11] using LINQ queries. It divides the transactional database into sub-datasets known as conditional patterns. Many threads ran concurrently on a multi-core computing system, one for each conditional pattern. They proved that the algorithm is faster by 2*x* and 4*x* times than the fast Eclat and FP-growth algorithms, respectively. A compressed bit matrix-based parallel algorithm for exploring frequent itemsets has been introduced by Zong-Yu et al., which uses both bottom-up and top-down approaches for efficient pruning [12]. It also uses OpenMP's parallel multithreaded, dynamic scheduling approach to extract frequent itemsets. Finally, they demonstrated that this approach reduces memory space, I/O overhead with a single database scan compared to the Apriori algorithm.

In [13], the authors have proposed a VDF approach for finding frequent itemsets using a Boolean matrix (FPMBM), where the presence of an item for the TID's is 1 and 0 for absence. It uses logical AND operation for finding support count from frequent 2-itemset to frequent *n*-itemsets until it is not empty. To control the number of iterations for candidate generation, it also uses additional information in the Boolean matrix, namely "number of iterations". Further, they demonstrated from the experiment that the FPMBM is efficient and more scalable than the existing ones.

Jen, T. Y., et al. have created a novel vertical format based parallel method for finding frequent patterns called Apriori_V with MapReduce platform. They proved that it provides a significant improvement in reducing the number of operations and decreasing computational complexity [14]. The authors in [15] have introduced a Parallel Regular Frequent Pattern (PRF) method to find out the regular-frequent patterns from large databases using VDF format and proved from the experiments that the algorithm reduced the number of database scans, I/O cost and inter-process communication.

In [16], the authors have reviewed the works related to Parallel Sequential Pattern Mining (PSPM), viz., partition-based, Apriori-based, pattern growth-based, and hybridized algorithms for PSPM. They also reviewed the open-source software's utilized in PSPM. Further, they summarized the issues and uses of PSPM in big data. In [17], the authors have proposed an FPM algorithm with a multi-core processor and Multiple Minimum Support called MMS-FPM. It quickly generated frequent patterns. It has been designed mainly to solve rare item problems. They have proved that the MMS-FPM is more superior to MSApriori and also scalable one. In [18], the authors have designed a Spark-based parallel Apriori algorithm called YAFIM (Yet Another Frequent Itemset Mining). The experimental result revealed that the proposed method is faster than the Apriori's MapReduce implementation by 18 times.

The existing literature found that no authors proposed parallel algorithms using a multithreaded approach with uni-processor systems. Thus, the research work focuses on a multithreaded approach with jagged array representation for VDF and novel pattern formation in finding frequent patterns, namely TB-NPF-VDF. It also compares the proposed work with the methods viz., Matrix-Apriori, VDF and NPF-VDF.

3. Proposed Methodology

The proposed work's main idea is to find frequent patterns for the transaction database TD. It contains four phases. Phase one scans TD first and converts it into VDF, in which a set of TIDs represents each item as in Eclat [19]. The second phase determines the frequent 1-itemset from VDF. The third phase sorts the frequent 1-itemset in ascending order based on the min_sup(δ) threshold, and it is stored in a matrix using the jagged array format. The δ of an itemset X is calculated by dividing the total transactions in which X occurs by the total number of transactions [20]. The fourth phase creates n-1 threads, one for each frequent 1-itemset except for the last one; where n represents the total items in frequent 1-itemset (L_1). Let $L_1=\{I_1, I_2,\ldots, I_n\}$, each thread generates frequent itemsets starting from frequent 2-itemset to frequent k-itemset until it is non-empty, where $k \geq 2$.

For finding frequent i-itemset, $i \geq 2$, each thread ($t_{x,1\leq x\leq n-1}$) uses the following procedure.

(1) When i=2, the thread forms the candidate patterns by combining I_x with I_{x+1} and finds the transactions in which the combination I_xI_{x+1} occur by intersecting the transactions in I_x and I_{x+1}. The item combinations whose support count $\geq \delta$ is selected as frequent i-itemset for item x.

(2) For i>2, each item in frequent (i-1)-itemset is combined with each frequent 1-itemset starting from the next item in the last item of frequent(i-1)-itemset and the transactions in which the combination exists is determined by intersecting the item in frequent (i-1)-itemset and the appropriate item in frequent 1-itemset. This procedure will be repeatedly performed as far as the frequent k-itemset is not null.

The proposed method uses multithreads and novel pattern formation with VDF to find frequent patterns is named TB-NPF-VDF. The main benefit of this method is that it generates fewer candidate itemsets than the classical Apriori and VDF because it avoids the items whose support count is lesser than the item at any instance of time for generating the patterns. As threads are used, the CPU is effectively utilized, and it is faster compared to processes. This method avoids checking the pattern for the Apriori property because the candidate patterns generated satisfies the Apriori property by default. Further, the time required for TB-NPF-VDF is less when compared to VDF. The memory requirement is minimized since the algorithm uses the matrix notation using a jagged array [8].

The algorithm for the proposed method is shown below, and the workflow of TB-NPF-VDF is illustrated in Fig.1.

TB-NPF-VDF: Algorithm to discover the frequent patterns

Input: TD - Transactional database;
δ - min_sup threshold;

Output: Frequent itemsets;

1: $vdf \leftarrow$ scan TD and store it in <$itemset$, TID_{list}> format;

2: $C_1 \leftarrow \emptyset$;

3: **for** each $item_i$ in vdf do

4: $SC \leftarrow$ count($TID_{list}(item_i)$); //determines the number of transactions in $item_i$

5: $C_1 \leftarrow C_1$.append ({$itemset$, TID_{list}, SC})// adds a row into C_1

6: **endfor**

7: **for** each $item_i$ in C_1 do

8: $L_1 \leftarrow \{item_i \mid SC(item_i) \geq \delta\}$

endfor
L_1←jagged(sort(L_1)) //sorts L_1 and converts it into a jagged matrix format
no_freq1_itemset←count(L_1) //determines the number of itemset in L_1
for (x=1; $x \leq$ (*no_freq1_itemset-1*); x++)
    t_x←create(thread) //create t_x for the $L_1[x]$
endfor
for each thread t_x **do**
    for (k=2; $L_k \neq \emptyset$; k++)
        if k==2 **then**
            new_pattern←<$I_x I_{x+1}$>;
            new_TID_list←Transactions(I_x)∩Transactions(I_{x+1});
        else if $k \geq 2$ **then**
            for each $item_j$ in L_{k-1} **do**
                new_item←last item in $item_j$
                new_pattern←{<$item_j I_y$>|I_y←next(*new_item*)}
                new_TID_list←Transactions($item_j$)∩Transactions(I_y);
            endfor
        endif
        SC←count(*new_TID_list*);
        C_k←C_k.append({*new_pattern*,*new_TID_list*});
        L_k←{C_k | $SC(C_k) \geq \delta$}
    endfor
endfor

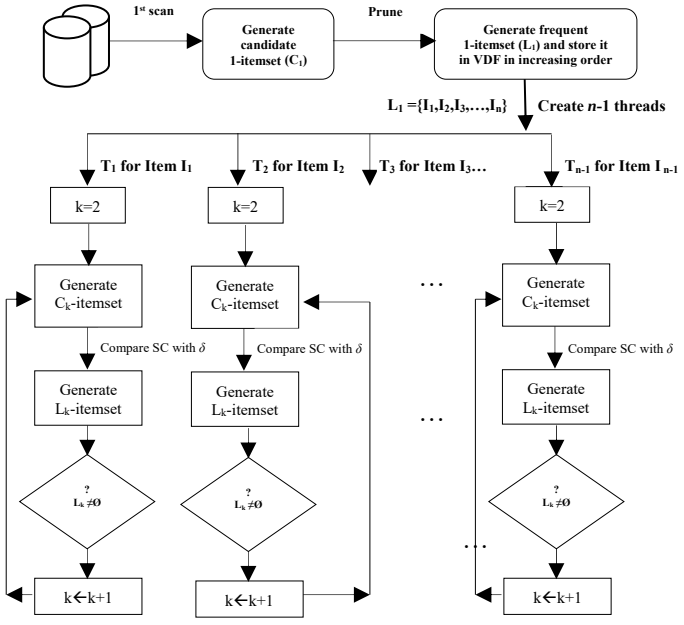

Fig. 1. Workflow of TP-NPF-VDF

3.1. *Example*

To understand the relevance of the proposed work, the Transactional Database (TD) shown in Table 1 has been considered. It consists of 12 items, namely A,B,C,D,E,F,G,H,I,K,M and P. The vertical representation of TD is shown in Table 2. Each row represents an item consisting of the item name and the TID's in which the item belongs. Assume the min_sup (δ) as 6. The candidate 1-itemset (C_1) consists of all the items in TD, the transaction IDs in which the items occurred, and the support count (*SC*), i.e. the total transactions in which the item appears. The C_1 for TD is shown in Table 3. Among them, the items viz., A,C,D,E,F,I,M and P satisfy the δ and form the frequent 1-itemset(L_1). The jagged array representation of the same is shown in Table 4 [21].

TID	Items Purchased
0	D,C,G,E,I,H,P,K,M
1	E,B,G,F,I,H,M,P
2	E,C,M
3	B,A,D,C,F,E,I,G,P
4	B,A,D,C,P,E
5	B,A,D,C,H,F,P
6	E,B,H,F,P,I,M
7	C,A,E,D,P,K,M
8	C,A,E,D,I,F,M,P
9	C,A,E,D,H,F,P,I,M

Table 1. Transactional Database (TD)

Item	Transaction ID's (TID's)
A	{3, 4, 5, 7, 8, 9}
B	{1, 3, 4, 5, 6}
C	{0, 2, 3, 4, 5, 7, 8, 9}
D	{0, 3, 4, 5, 7, 8, 9}
E	{0, 1, 2, 3, 4, 6, 7, 8, 9}
F	{1, 3, 5, 6, 8, 9}
G	{0, 1, 3}
H	{0, 1, 5, 6, 9}
I	{0, 1, 3, 6, 8, 9}
K	{0, 7}
M	{0, 1, 2, 6, 7, 8, 9}
P	{0, 1, 3, 4, 5, 6, 7, 8, 9}

Table 2. Transactional Database in VDF

C_1

Itemset	TID's	SC
A	{3, 4, 5, 7, 8, 9}	6
B	{1, 3, 4, 5, 6}	5
C	{0, 2, 3, 4, 5, 7, 8, 9}	8
D	{0, 3, 4, 5, 7, 8, 9}	7
E	{0, 1, 2, 3, 4, 6, 7, 8, 9}	9
F	{1, 3, 5, 6, 8, 9}	6
G	{0, 1, 3}	3
H	{0, 1, 5, 6, 9}	5
I	{0, 1, 3, 6, 8, 9}	6
K	{0, 7}	2
M	{0, 1, 2, 6, 7, 8, 9}	7
P	{0, 1, 3, 4, 5, 6, 7, 8, 9}	9

Table 3. Candidate 1-Itemset

L_1

1-Itemset	TID's								
A	3	4	5	7	8	9			
C	0	2	3	4	5	7	8	9	
D	0	3	4	5	7	8	9		
E	0	1	2	3	4	6	7	8	9
F	1	3	5	6	8	9			
I	0	1	3	6	8	9			
M	0	1	2	6	7	8	9		
P	0	1	3	4	5	6	7	8	9

Table 4. Jagged Array Representation of L_1

To generate fewer candidate itemsets, this research work uses a novel pattern generation method rather than the natural join used in the Apriori algorithm. For that, the L_1 is sorted in ascending order based on *SC* and replaced with L_1 as illustrated in Table 5.

The sorted L_1 contains 8 items, and this work creates 7 threads because the frequent 1-itemset contains 8 items. Thread-1 is for the item <A>, Thread-2 is for item <B>, etc. The Thread-1 first generates the following patterns.

<AF>, <AI>, <AD>, <AM>, <AC>, <AE> and <AP> and for each pattern, set intersection is calculated by using the TID's in each item of the pattern. For example, for the pattern <AF> the set intersection is calculated as {3, 4, 5, 7, 8, 9} ∩ {1, 3, 5, 6, 8, 9} = {3, 5, 8, 9} and *SC*=4.

L_1

1- Itemset	TID's								
A	3	4	5	7	8	9			
F	1	3	5	6	8	9			
I	0	1	3	6	8	9			
D	0	3	4	5	7	8	9		
M	0	1	2	6	7	8	9		
C	0	2	3	4	5	7	8	9	
E	0	1	2	3	4	6	7	8	9
P	0	1	3	4	5	6	7	8	9

Table 5. Sorted L_1

Similarly, the *SC* for other patterns viz., <AI>, <AD>, <AM>, <AC>, <AE> and <AP> is calculated as stated above. The patterns whose $SC \geq \delta$ will be considered as the frequent 2-itemset for the item <A> and are represented in Table 6. For this case, the patterns <AD>, <AC> and <AP> satisfies the δ.

Item	TID's					
<AD>	3	4	5	7	8	9
<AC>	3	4	5	7	8	9
<AP>	3	4	5	7	8	9

Table 6. Frequent 2-Itemset for <A> by Thread-1

Next, the method generates the candidate 3-itemsets for each frequent 2-itemset in Table 6 as follows.

(1) For the frequent 2-item <AD>, the items viz., <M>, <C>, <E> and <P> are considered from frequent 1-itemset because <M> is the next item after <D> where, <D> is the last item in frequent 2-itemset <AD>. The patterns generated are <ADM>, <ADC>, <ADE> and <ADP> and for them, the transactions in which the pattern occurs and *SC* is calculated as follows.
From Table 6, the TID's of <AD> is {3, 4, 5, 7, 8, 9} and from Table 5 the TID's of <M> is {0, 1, 2, 6, 7, 8, 9}. Therefore, {3, 4, 5, 7, 8, 9} ∩ {0, 1, 2, 6, 7, 8, 9} = {7, 8,T} and *SC*=3. Similarly, for <ADC>, <ADE> and <ADP> is also calculated.

(2) For the frequent 2-item <AC>, the items from <E> i.e. <E> and <P> are considered. The patterns generated are <ACE> and <ACP> and *SC* is calculated as above.

(3) For the frequent 2-item <AP>, there is no candidate 3-itemset because there is no next item after <P>.

The candidate 3-itemset generated by Thread-1 are <ADM>, <ADC>, <ADE>, <ADP>, <ACE> and <ACP>. Among them the patterns viz., <ADC>, <ADP> and <ACP> satisfies δ forms frequent 3-itemset and represented by Table 7.

Itemset	TID's					
<ADC>	3	4	5	7	8	9
<ADP>	3	4	5	7	8	9
<ACP>	3	4	5	7	8	9

Table 7. Frequent 3-Itemsets for <A> By Thread-1

The frequent 3-itemset for <A> is not empty, so the method generates the candidate 4-itemset. They are <ADCE> and <ADCP>. The TID's for <ADCE> is calculated as {3, 4, 5, 7, 8, 9} ∩ {0, 1, 2, 3, 4, 6, 7, 8, 9}={3, 4, 7, 8, 9} and *SC* of <ADCP> is 5. Similarly, for <ADCP>, the TID's are {3, 4, 5, 7, 8, 9}∩{0, 1, 3, 4, 5, 6, 7, 8, 9} = {3, 4, 5, 7, 8, 9}. The SC of <ADCP> is 6 and it is illustrated in Table 8.

Itemset	TID's					
<ADCP>	3	4	5	7	8	9

Table 8. Frequent 4-Itemsets for <A> by Thread-1

Now, candidate 5-itemset for the item <A> is Ø. So Thread-1 stops its execution and returns <AD>, <AC>, <AP>, <ADC>, <ADP>, <ACP> and <ADCP> as frequent items for <A>. Similarly, the other threads generate frequent itemsets for other frequent 1-itemset in parallel as shown from Table 9 to Table 19.

Itemset	TID's					
<FP>	1	3	5	6	8	9

Table 9. Frequent 2-Itemset for <F> by Thread-2

Itemset	TID's					
<IE>	0	1	3	6	8	9
<IP>	0	1	3	6	8	9

Table 10. Frequent 2-Itemset for <I> by Thread-3

Itemset	TID's					
<IEP>	0	1	3	6	8	9

Table 11.frequent 3-Itemset for <I> by Thread-3

Itemset	TID's						
<DC>	0	3	4	5	7	8	9
<DE>	0	3	4	7	8	9	
<DP>	0	3	4	5	7	8	9

Table 12. Frequent 2-Itemset for <D> by Thread-4

Itemset	TID's						
<DCE>	0	3	4	7	8	9	
<DCP>	0	3	4	5	7	8	9
<DEP>	0	3	4	7	8	9	

Table 13. Frequent 3-Itemset for <D> by Thread-4

Itemset	TID's					
<DCEP>	0	3	4	7	8	9

Table 14. Frequent 4-Itemset for <D> by Thread-4

Itemset	TID's						
<ME>	0	1	2	6	7	8	9
<MP>	0	1	6	7	8	9	

Table 15. Frequent 2-Iemset for <M> By Thread-5

Itemset	TID's					
<MEP>	0	1	6	7	8	9

Table 16. Frequent 3-Itemset for <M> by Thread-5

Itemset	TID's						
<CE>	0	2	3	4	7	8	9
<CP>	0	3	4	5	7	8	9

Table 17. Frequent 2-Itemset for <C> by Thread-6

Itemset	TID's					
<CEP>	0	3	4	7	8	9

Table 18. Frequent 3-Itemset for <C> by Thread-6

Itemset	TID's							
<EP>	0	1	3	4	6	7	8	9

Table 19. Frequent 2-Itemset for <E> by Thread-7

Table 20 depicts the candidate and frequent items, the total number of candidates and frequent items generated by the TB-NPF-VDF for the given TD. The total number of candidate items generated using TB-NPF-VDF is 56, and it is less when compared to VDF.

Itemset	Candidate Items	Total[#]	Frequent Items	Total[$]
1-itemset	{A,B,C, D, E, F,G, H, I, K, M,P,M}	13	{A,C,D,E,F, I,M, P}	8
2-itemset	{AF,AI,AD,AM,AC,AE,AP,FI,FD,FM,FC,FE,FP,ID,IM, IC,IE,IP,DM,DC,DE,DP,MC,ME,MP,CE,CP,EP}	28	{AD,AC,AP,FP,IE,IP,DC,DE,DP, ME,MP,CE,CP,EP}	14
3-itemset	{ADM,ADC,ADE,ADP,ACE,ACP,IEP,DCE,DCP,DEP, MEP, CEP}	12	{ADC, ADP, ACP, IEP,DCE, DCP, DEP,MEP, CEP}	9
4-itemset	{ADCE, ADCP,DCEP}	3	{ADCP, DCEP}	2
Total		**56**		**33**

[#]Number of candidate items [$]Number of frequent items

Table 20. Details of Itemsets for TD

4. Experimental Results and Discussion

The algorithms viz., Matrix-Apriori, VDF, NPF-VDF and TB-NPF-VDF were implemented using the Python programming language (version 3.8.2). To estimate the performance of TB-NPF-VDF, the research work used four real-time datasets downloaded from the FIMI repository and an open-source Data Mining Library. Table 21 describes the characteristics of datasets. The purpose of using these datasets is that they have been used as a reference by researchers primarily for FPM and ARM-based research. To do a uniform and fair comparison, the experiments for all the datasets of all algorithms were conducted using the same software and hardware configurations. The experiments were performed using 8.00GB RAM, Intel Core i7 with 2.40GHz 64-bit processor and Windows 8.1. All algorithms' runtime performance (Matrix-Apriori [7], VDF, NPF-VDF, TB-NPF-VDF) for the four datasets with different min_sup percentages ranging from 20% to 70% were tabulated in Table 22.

Datasets	Transaction count	Item count	Average item count/transaction
chess	3196	75	37.00
mushrooms	8416	119	23.00
T25i10d10k	9976	929	24.77
c20d10k	10000	192	20.00

Table 21. Characteristics of Datasets

min_sup (%)	Runtime (in Sec.)			
	Matrix -Apriori	VDF	NPF-VDF	TB-NPF-VDF
chess				
20	20.7578	16.8578	13.3578	6.5267
30	19.6365	16.0452	12.1455	5.0325
40	17.7750	14.0750	10.0720	4.5635
50	16.3028	13.3017	9.0017	3.2634
60	15.3625	12.7943	8.2934	2.4571
70	14.8546	11.9825	7.4822	2.0012
mushroom				
20	23.2135	21.1215	18.0016	12.1024
30	21.3426	20.0462	17.0642	11.5642
40	20.0035	19.7083	14.1038	10.7869
50	19.2002	18.2058	13.2044	10.0063
60	18.0805	17.7898	12.7240	8.5698
70	17.5652	15.9575	11.4530	7.9586
t25i10d10k				
20	25.2145	23.3254	20.3325	15.1267
30	23.9625	21.4578	19.4258	13.9568
40	21.5467	20.0025	17.9857	12.0127
50	20.3859	18.7621	16.2456	11.6321
60	19.5321	18.0056	15.0012	10.5212
70	18.4521	16.0527	13.7564	9.2451
c20d10k				
20	26.0014	24.4253	22.8342	17.7586
30	24.9532	22.6752	21.5062	15.9802
40	22.4251	21.9546	20.0412	13.7542
50	21.5621	19.4316	18.8562	11.9892
60	20.1425	19.0012	17.0124	11.0016
70	19.1478	17.5242	15.9351	10.0142

Table 22. Performance Results

Figures 2 to 5 show the graphical representation of the runtime comparison between the algorithms viz., Matrix-Apriori, VDF, NPF-VDF and TB-NPF-VDF for the datasets, namely chess, mushroom, t25i10d10k and c20d10k, respectively. From Table 22 and from figures 2 to 5, it was observed that the runtime performance of TB-NPF-VDF outperforms than Matrix-Apriori, VDF and NPF-VDF. On an average, the runtime performance is improved from 20.3092 to 9.9094.

Further, to prove statistically, a Welch two-sample *t*-test is being performed between the runtimes of Matrix-Apriori and TB-NPF-VDF. The test was done to determine whether the mean runtimes of Matrix-Apriori and TB-NPF-VDF are equal to each other or not. The null hypothesis is taken as that the two mean runtimes are equal, and the alternative is that they are not equal. The test is performed using the R tool for each dataset, and the results are tabulated in Table 23.

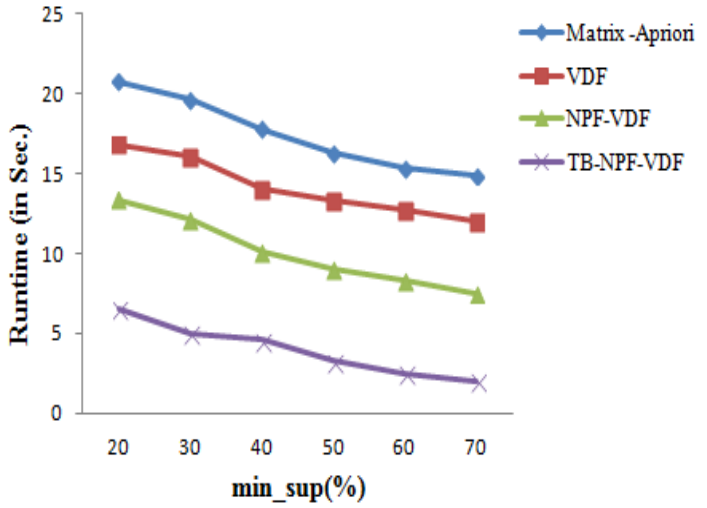

Fig. 2. The execution time of Matrix-Apriori, VDF, NPF-VDF and TB-NPF-VDF for chess dataset

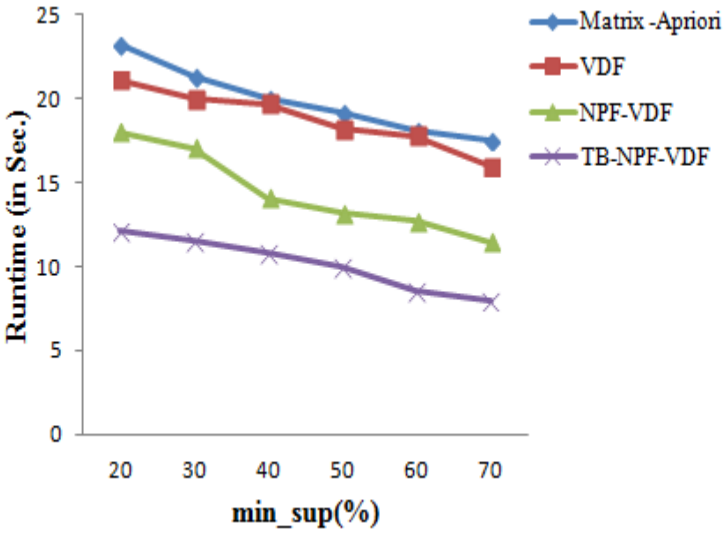

Fig. 3. The execution time of Matrix-Apriori, VDF, NPF-VDF and TB-NPF-VDF for mushroom dataset

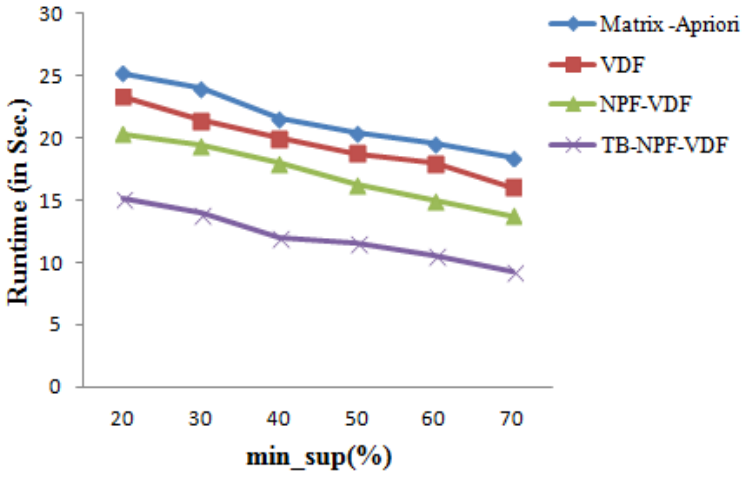

Fig. 4. The execution time of Matrix-Apriori, VDF, NPF-VDF and TB-NPF-VDF for t25i10d10k dataset

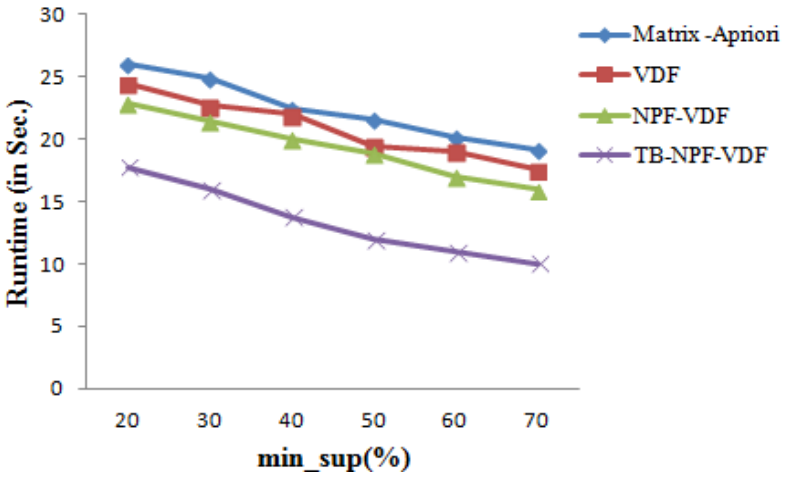

Fig. 5. The execution time of Matrix-Apriori, VDF, NPF-VDF and TB-NPF-VDF for c20d10k dataset

Dataset	p-value
chess	1.207×10^{-06}
mushroom	6.785×10^{-06}
t25i10d10k	5.611×10^{-05}
c20d10k	0.0002914

Table 23. Results of *t*-Test

From the observation of *t*-test results, it is noted that for all datasets, the p-value is ≤ 0.05 (5%) which concluded that the two means are not equal, which means that there are significant differences between the runtimes. Therefore, the proposed method TB-NPF-VDF is more efficient in terms of runtime than the others.

The reason for enhancing the performance is that the concurrent execution of the tasks using a multithreaded approach speeds applications up and reduced the time required for execution by utilizing the CPU effectively. With novel pattern generation, the set of candidate elements generated is less than the existing ones. Further, it scans the database only once during the entire process.

5. Conclusion

Many FPM algorithms were introduced in the field of data mining. Each algorithm has its own merits and demerits and is unsuited for all real-life situations. A new approach called TB-NPF-VDF has been introduced in this research article to discover the frequent patterns that efficiently combine the power of VDF, NPF, and multithread concepts. Experiments were carried out on real-time datasets using python implementation for the existing and proposed methods. TB-NPF-VDF has been proven to be superior to other sequential approaches through memory usage and run time. The main advantage is that it discovers frequent patterns with less time and saves memory with jagged array representation for the VDF matrix. In future, the work can be improved by applying new and efficient optimization techniques.

References

[1] Guo, Y. M.; Wang, Z. J. (2010): A vertical format algorithm for mining frequent item sets. Proceedings of 2nd International Conference on Advanced Computer Control (IEEE Xplore), **4**, pp. 11-13.
[2] Han, J.; Kamber, M.; Pei, J. (2011): *Data mining concepts and techniques*, 3rd edn. Morgan Kaufmann.
[3] Aqra, I.; Herawan, T.; Ghani, N. A.; Akhunzada, A.; Ali, A.; Razali, R. B.; Choo, K. K. R. (2018): A novel association rule mining approach using TID intermediate itemset. PloS one, **13**(1), pp. 01-32.
[4] Subhashini, A.; Karthikeyan, M. (2019): Itemset Mining using Horizontal and Vertical Data Format, International Journal for Research in Engineering Application & Management. **5**(3) pp. 534-539.
[5] Gawwad, M. A.; Ahmed, M. F.; Fayek, M. B. (2017): Frequent itemset mining for big data using greatest common divisor technique. Data Science Journal, **16**(25), pp. 1-10.
[6] Usha, D.; Rameshkumar, K. (2014): A Complete Survey on application of Frequent Pattern Mining and Association Rule Mining on Crime Pattern Mining. International Journal of Advances in Computer Science and Technology, **3**(4), pp. 264-275.
[7] Pavón, J.; Viana, S.; Gómez, S. (2006): Matrix Apriori: Speeding up the Search for Frequent Patterns. Databases and Applications, pp. 75-82.
[8] Sumathi, P.; Murugan, S. (2018): A Memory Efficient Implementation of Frequent Itemset Mining with Vertical Data Format Approach. International Journal of Computer Sciences and Engineering, **6**(11), pp. 152-157.
[9] Chon, K.W.; Hwang, S. H.; Kim, M. S. (2018): GMiner: A fast GPU-based frequent itemset mining method for large-scale data. Information Sciences, **439**, pp. 19-38.

[10] Huang, Y. S.; Yu, K. M.; Zhou, L. W.; Hsu, C. H.; Liu, S. H. (2013): Accelerating parallel frequent itemset mining on graphics processors with sorting. Proceedings of IFIP International Conference on Network and Parallel Computing, pp. 245-256.
[11] Huang, C. H.; Leu, Y. (2015): A LINQ-based conditional pattern collection algorithm for parallel frequent itemset mining on a multi-core computer. Proceedings of ASE BigData & Social Informatics, pp. 1-6.
[12] Zong-Yu, Z.; Ya-Ping, Z. (2012): A parallel algorithm of frequent itemsets mining based on bit matrix. Proceedings of IEEE International Conference on Industrial Control and Electronics Engineering, pp. 1210-1213.
[13] Tanna, P.; Ghodasara, Y. (2015): Analytical Study and Newer Approach towards Frequent Pattern Mining using Boolean Matrix. IOSR Journal of Computer Engineering, **17**(3), pp. 105-109.
[14] Jen, T. Y.; Marinica, C.; Ghariani, A. (2016): Mining frequent itemsets with vertical data layout in MapReduce. Proceedings of International Workshop on Information Search, pp. 66-82.
[15] Vijay Kumar, G.; Valli Kumari, V. (2013): Parallel Regular-Frequent Pattern Mining in Large Databases. International Journal of Scientific & Engineering Research, **4**(6).
[16] Gan, W.; Lin, J. C. W.; Fournier-Viger, P.; Chao, H. C.; Yu, P. S. (2019): A survey of parallel sequential pattern mining. ACM Transactions on Knowledge Discovery from Data (TKDD), **13**(3), pp. 1-34.
[17] Huynh, B.; Trinh, C.; Dang, V.; Vo, B. (2019): A parallel method for mining frequent patterns with multiple minimum support thresholds, International Journal of Innovative Computing. Information and Control, **15**(2), pp. 479-488.
[18] Qiu, H.; Gu, R.; Yuan, C.; Huang, Y. (2014): YAFIM: a parallel frequent itemset mining algorithm with spark. Proceedings of IEEE International Parallel & Distributed Processing Symposium Workshops, pp. 1664-1671.
[19] Shruti, I.; Abhay, K. (2018): Parallel Eclat with Large Data Base Parallel Algorithm and Improve its Effectiveness. International Journal of Engineering Trends and Technology, **60**(3), pp. 180-183.
[20] D. Kalpana, Data Mining Apriori Algorithm Implementation Using R, International Research journal of Engineering and Technology. **4**(11), pp. 1810- 1815.
[21] Sumathi, P.; Murugan, S. (2021): GNVDF: A GPU-accelerated Novel Algorithm for Finding Frequent Patterns Using Vertical Data Format Approach and Jagged Array. International Journal of Modern Education and Computer Science (IJMECS), **13**(4), pp. 28-41.

Authors Profile

P.Sumathi received her B.Sc and M.Sc degrees in Computer Science from Seethalakshmi Ramaswami College (affiliated to Bharathidasan University), Tiruchirappalli, India in 2001 and 2003 respectively. She received her M.Phil degree in Computer Science in 2008 from Bharathidasan University. She is presently working as an Assistant Professor in the Department of Computer Science, Vysya College, Salem. She is currently pursuing Ph.D, a degree in Computer Science in Bharathidasan University. Her research interests include Data Mining, Data structures and Database concepts.

S.Murugan received his M.Sc degree in Applied Mathematics from Anna University in 1984 and M.Phil degree in Computer Science from Regional Engineering College, Tiruchirappalli in 1994. He is an Associate Professor in the Department of Computer Science, Nehru Memorial College (Autonomous), affiliated to Bharathidasan University since 1986. He has 32 years of teaching experience in the field of Computer Science. He has completed his Ph.D degree in Computer Science with a specialization in Data Mining from Bharathiyar University in 2015. His research interest includes Data and Web Mining. He has published more than 25 research articles in reputed National and International journals.

V.Umadevi obtained her M.Sc degree in Computer Science & Information Technology and M.Phil degree in Computer Science from Madurai Kamaraj University. She has completed her Ph.D degree in Computer Science from CMJ University. Besides, she has received M.Tech and MBA degrees. She has 15 years of teaching experience in Computer Science. Her area of teaching and research interests include Management Information Systems, Project Management and Wireless Sensor Networks. She has published 28 research papers in National and International journals and authored three books. Also produced one Ph.D candidate. She has received National Award for "South Indian Achiever" in March 2020 and a "Lifetime Achiever" award from International Lions Club in March 2021. She has published a patent entitled "AI abetted material synthesising for hybrid metal rubber composite and 3D Printing" in August 2021.