International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-5, March 2020

Frequent Itemset Mining in a Unique Scan using Transaction Database

A. Subashini, M. Karthikeyan

Abstract: ***In recent year, frequent Itemset Mining (FIM) has occurred as a vital role in data mining tasks. The search of FIM in a transactions data is discovered in this paper, pull out hidden pattern from transactions data. The main two limitation of the Apriori algorithm are undertaken, first, its scans the complete Databases at every passes to compute the supports of every itemset produced and secondly, the user defined responsive to variation of min_sup (minimum supports) thresholds. In this paper, proposed methodology called frequent Itemset Mining in unique Scan (FIMUS), needs a scan only one time of transaction databases to extract frequent itemsets. The generation of a static numbers of candidate Itemset is an exclusive feature, individually from the threshold of min_sup, which reduces the execution time for huge database. The proposed algorithm FIMUS is compared with Apriori algorithm using benchmark database for a dense databases. The experimental result confirms the scalability of FIMUS.***

Keywords**: *Frequent itemset mining, itemset mining, unique scan, Apriori, min_sup.***

I. INTRODUCTION

The aim of frequent Itemset Mining (FIM) is to extract high support itemsets from a huge transaction databases. Let T_{DB} be a set of *n* transaction,$\{T_1,T_2,\ldots,T_n\}$ is a transactions databases then I_s a set of m various item $\{I_1,I_2,\ldots I_m\}$. An Itemsets Z is a subset of the set of items ($Z \subseteq I_s$).

Support (Z) $= \frac{number\ of\ transaction\ that\ contain\ Z}{no\ of\ all\ transation\ in\ T_{DB}}$,

If itemset Z support is not less than a min_sup threshold given by user is frequent Item (FI) [1] otherwise itemset Z is frequent item (FI) .There are two Approaches proposed for solving FIM problem. First Apriori [1], which produce $i-size$ candidate itemset from the $(i-1)-size$ FIs then check the frequency of the candidate generation itemset. Second approach based on tree based i.e. FP-Growth [2]. It compress database in memory using a tree structure then it implement recursive process to discover FIs. Compared to Apriori, FP-Growth reduces scanning the database. At the same time these both methods consume huge memory, mainly while dealing with huge databases.

In this paper we proposed a novel methodology called FIMUS (Frequent Itemset Mining in a Unique Scan), it solve the FIs mining problems with a unique scan of database T_{DB}. In FIMUS, candidate itemset is produced first from every transaction and stores in the hash tables to keep data with their support. While next transaction is generated, it increments the count of items which items are already exist in the hash table. Else, a new input entry counter is initialized. At final, an Itemsets frequency existence in the hash table is compared with min_sup of items, to find which items wants to retain (frequent). This proposed method have been tested on various FIM instances. The experimental result shows the efficiency of FIMUS compared with Apriori for normal and huge database with multiple min_sup.

II. RELATED WORKS

The strategic for explaining the FIM (Frequent itemset mining) problematic can be separated into two kinds.

(i) To *generate and checks frequency* strategy, i.e. first items are generated and then checks the frequency.

(ii) The *divide and conquer* strategy. (i.e.) a tree structure approach, it compress the transaction database and then FIs extract by applying the mining process recursively.

In the following, we discuss more about FIM approaches on two kinds. The very first algorithm **Apriori**, by Agrawal et al. [1] to generate and checks the frequency threshold. Here candidate generate $i-size$ itemset, then the frequent $(i-1)-size$ itemset are calculated and combined. This procedure repeats till null candidates obtained in an iterations. The next algorithm **Dynamic itemset counting (DIC)** proposed by S Brin et.al. [4], generalize of algorithm Apriori where the data divided into x equal size so fits in memories. DIC then folds supports of one itemset for the 1st partitions. Frequent item found locally are used for 2-size itemset and then 2nd partition is to find supports of every candidates. Process repeats for remaining partitions. It terminates when there is no generation of candidates from the existing partitions and counts every previous candidates. Another algorithm **Eclat**, proposed by Zaki et al.[8]. This algorithm use vertical transaction ID list of itemset. $i-size$ Frequent itemset are ordered into disjoints correspondence class by common $(i-1)-size$ prefix, thus $(i+1)-size$ candidate itemset can be generate by joining set of frequent $i-size$ itemset from similar class. Then by intersecting the tid-list the support of candidate itemset can be calculated. In [9], to store and compress the transaction in a tid-list, a data structured is proposed. In this structure it reduces the number of transaction scan. But frequent itemset only extract.

Revised Manuscript Received on March 30, 2020.
* Correspondence Author
A.Subashini*, Assistant Professor in the Department of Computer Application, Government Arts College, C.Mutlur, Chidambaram, Tamil Nadu, India.
M.Karthikeyan, Assistant Professor in the Division of Computer and Information Science, Annamalai University, Annamalai Nagar, Chidambaram, Tamil Nadu, India.

For the second kind of divide and conquer strategy the **FP-Growth** algorithm [2] is used to compress tree (FP-tree) structure without candidate generation itemset mining for a Whole set of frequent itemset. It partition into two stages:

. Constructing a FP-tree by reading the DB and map every transaction on that paths in the FP-tree, and concurrently count each item supports.

i. All Frequent itemset extracts straight from FP-tree using the strategy bottom-up to discover every possible FIs that ends with a particular itemset.

Next algorithm proposed by Cerf et al. [10] has proposed the **NFP-growth** algorithms. It improve the existing FP-growth by constructing an individual header tables that allow to create a frequent patterns tree only once. So this increases the process speeds. The Proposed algorithm New FP-Growth for mining uncertain data [11]. For storing the uncertain data it progress a tree, which occur counts of nodes in the summation of count of every child node. Which permit to counts the supports of all candidate itemsets. Mining various type of FIs, Maximal FIs, closed FIs and Categorical FIs using FP-array technique which reduce the traverse FP-tree is proposed [12]. A survey of FIM algorithm found in [13].

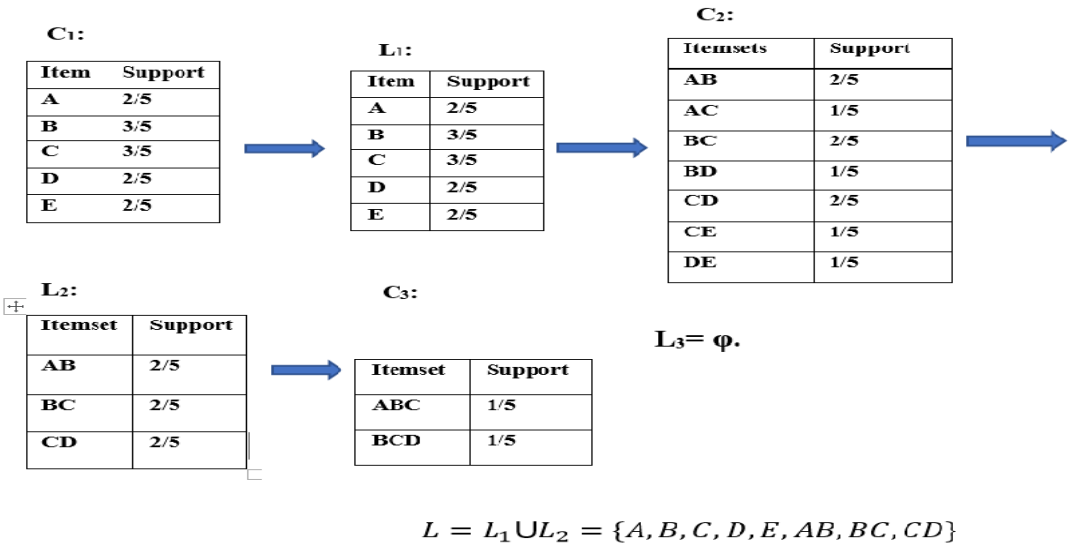
The first kind of generate and check frequency strategy needs many scans the database to generate every FIs, while the divide and conquer needs two scans only the database but high memory to compressing the database into a tree. It make the divide and conquer approach inefficient for huge transactional database. Recent approach has propose the number of scans reduce the transaction DB. I.e. BSO-ARM [14], PGARM [15] and PeARM [16] etc., .This approach deals with the FIM in time. But the limited quality of mining, i.e., it discovers only a parts of FIs and it miss numerous items.

III. APRIORI HEURISTIC METHODOLOGY

The main aim of Apriori algorithm is to minimize the space of search of frequent and rare Itemset by travelling iteratively the candidate itemset. In Apriori based algorithms, an itemsets is in i size are frequent if and only if every subset are frequent. Otherwise, it consider as Infrequent. Because, at all iteration i, the candidate itemset size i is generate by join two frequent itemset size (i-1). Repeating the above process until the candidates' itemset size i was unfilled. To define the frequent itemset at every iterations of the candidate and their supports of all candidate Itemsets are figured. If min_sup was not less than the threshold then it is added to frequent itemset. The support of every itemset z, was calculate as the relation between the sums of transaction contain z, and the whole transaction T_{DB} sum. To work out the support of z, the whole database T_{DB} is scanning, so z is tested against every transactionT_{DB} i . If $z \in T_{DB}$ then frequency ratio is increased by one in the numerator. Following example consist with five transaction and five Itemsets {A, B, C, D, E}, as given in Table 1.

Table 1. Example of a transaction database

TID	Items
T1	{A, B}
T2	{B, C, D}
T3	{A, B, C}
T4	{E}
T5	{C, D, E}



$$L = L_1 \cup L_2 = \{A, B, C, D, E, AB, BC, CD\}$$

Fig1. Illustrate of Apriori algorithm

In Fig1. Explains the result of the Apriori algorithms with input (minimum support) ɱ*sup* set to 0.4. To calculate support of every Itemsets of size 1, the transaction database was first scanned and extracted the frequent item of size 1. Example shows that every candidate item is frequent because of support exceeds 0.4, so there is no infrequent item in size 1. Next iteration, the frequent itemset of size one is joined to extract size 2 candidate itemset, we got frequent item {*AB, BC, CD*}. Then itemset {*ABC, ABD, BCD*} are candidate of Size three, then its supports is not lesser than 0.4 is consider as frequent itemset. The set of frequent itemset with minimum

Support no less than 40% is intersection of frequent itemset of size 1 and size 2, i.e. {A, B, C, D, E, AB, *BC, CD}.*

The Apriori limitations:

1. Numerous scanning of the transaction databases is essential: To work out the supports of candidates' itemset, every present approach is built on the Apriori algorithm which scans the whole transaction databases. Therefore, the numeral scan of databases is relative to the numeral of generated candidates' itemset, that which tend to be higher for larger database.

2. Setting min_sup user threshold was challenging: The Apriori experimental is very complex when the min_sup is varied. While choosing low minimum support, very high number of candidate itemset is found, which degenerates the executions time of the algorithm, as every candidate itemsets needs a scans of the whole databases.

III. FREQUENT ITEMSET MINING IN A UNIQUE SCAN (*FIMUS*)

In this division we present our proposed algorithms, Frequent Itemset Mining in Unique Scan (FIMUS). This algorithm explanation is following by a theoretic analyses of FIMUS is compared to Apriori algorithm.

A. *FIMUS Algorithm Methodology*

The main scope of *FIMUS* is discovering frequent itemset with only one scanning the database and also reduces the candidate generation. Hence to overcome the Apriori algorithm limitation. The important idea of *FIMUS* is to produce every possible itemset for every transactions. If itemset *z* have previously produced then it increments its support, else its supports is creates and initialized to one. This processing is recurring till whole transaction has processed. *FIMUS* agrees to find every frequent itemset in a single scan of database. *FIMUS is* complete, since the frequent Itemsets is extract straight from the transaction databases and a set of certain itemsets is frequent itemset iff it was find ($\eta sup \times n$) time from transactional database, where ɳsup is minimum support and n is number of customer transaction. Accordingly, there is no information lost in the process of Itemsetsgeneration.

FIMUS Algorithm explains in brief. Transaction database T_{DB} , is input for algorithm *FIMUS*, and ɳ*sup*, minimum support. Internal data structures representation is also used by a hashtable H_T to stock every itemset which is generated with occurrence of their partial number. The set of every frequent Itemsets *FI* is return as output in this algorithm.

First, the set of itemset,I_s, is work out from every transaction in T_{DB}. For example, if the transactions $T_{DB}\, i$ contains the items *A, B, and C*, then I_s contain the Itemsets A, B, C, AB, AC, BC, ABC. Later, every Itemsets, $z \in I_s$, is saved in the hashtable H_T. If z already exist as a key inH_T, then the entry with key z in ħ, i.e., $H_T(z)$ is increased by one. Else, a new entry with key z, is created in H_T and initialize as one. In Final, every entry, $z \in H_T$ with supports exceeds the minimum supports *ɳsup* is added to the set of the frequent itemsets *FI*.

In Fig 2. Show the FIMUS algorithms executed using the example of Table 1 with *ɳsup* sets as 0.4. FIMUS start scan the first transactions *{A, B}* then extract from its every possible candidate itemset, i.e., *{A, B, AB}*. In the first stage, hashtable H_T was empty, for every candidate itemset first transaction entry in hashtable H_T was created and count one is initialized. For transaction T2 {*B, C, D*}, FIMUS determine every possible likely candidates itemset, that are {*B, C, D, BC, BD, CD, BCD*}.

From those Itemsets {B} is already exists in hashtableH_T, therefore it is increment by one, else remaining entire candidate itemset was created then initialize as one. Remaining transaction repeats the same procedure. At the final, the itemset in H_T supports are not less than 0.4 is selected. The output of a sets of frequent itemset is{ A,B,C,D,E,AB,BC,CD}.

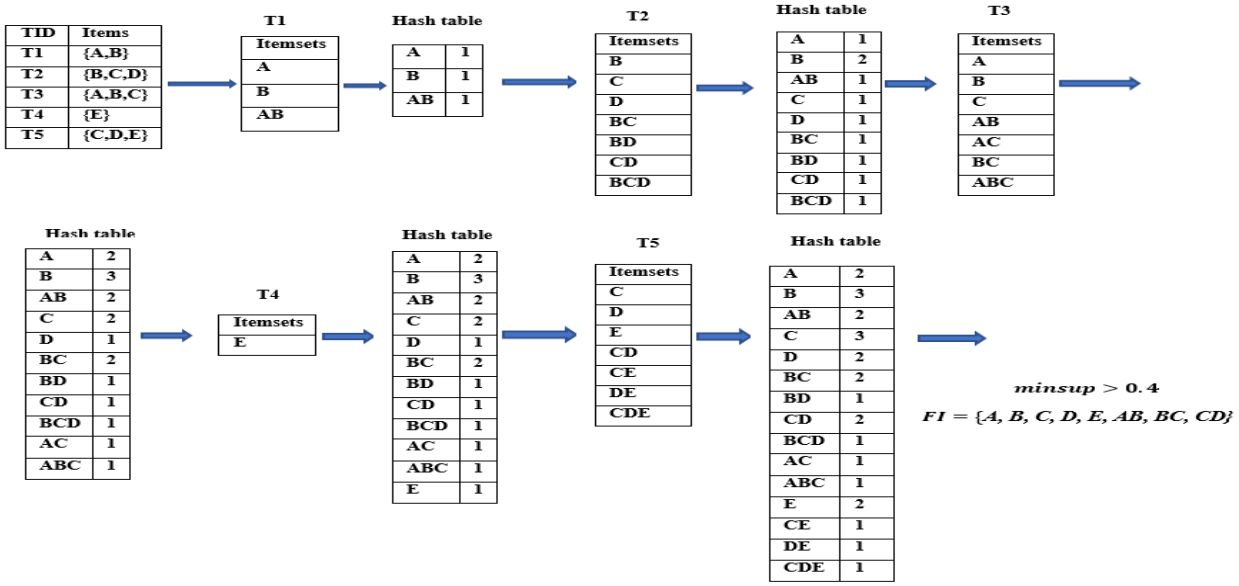


Fig 2. FIMUS approach illustration {A, B, C, D, E, AB, BC, CD}

Algorithm: *FIMUS Algorithm*

Input: T_{DB} *transactional database and user defined threshold* ηsup.
Output: ***FIs*** *set of frequent itemsets.*
For *every* $T_{DB}\,i$ *transactions* ***do***
$I_s \leftarrow$ *Gen_All_Itemsets* ($T_{DB}\,i$)
for *each itemsets* $z \in I_s$ ***do***
if $z \in \mathrm{H}_T$ ***then***
$\mathrm{H}_T(t) \leftarrow \mathrm{H}_T(t) + 1$.
else
$\mathrm{H}_T(t) \leftarrow 1$.
end if
end for
end for
FIs $\leftarrow \emptyset$.
for *itemsets* $Z \in \mathrm{H}_T$ ***do***
if $\mathrm{H}_T(t) \geq \eta sup$ ***then***
FIs $\leftarrow$ *FIs* $\cup z$
end if
end for
return *FIs*

B. *Theoretical Analysis:*

The execution time of FIMUS is the summation of (i) the rate of generating itemsets, and (ii) determining the rate of frequent itemset. Concerning the previous, the numbers of every candidate generates from the transactions $T_{DB}\,i$ is $2^{|T_{DB}\,i|} - 1$, where $|T_{DB}\,i|$ represent the numbers of item of $T_{DB}\,i$. The entire numbers of generating candidate itemset is so $\sum_{i=1}^{n} 2^{|T_{DB}\,i|} - 1$, where n is the numbers of transaction in database T_{DB} . If q is max numbers of item generating per transactions, so the numbers of candidate itemset is at most $n(2^q - 1)$. the difficulty of operation need for a generations of itemset is than $O\big(n(2^q - 1)\big)$. for defining the frequent itemset, the hashtable have been scanned for every candidate itemsets, using minimum support ηsup for evaluating its frequency. This process was $O\big(n(2^q - 1)\big)$.

Thus, the execution time of FIMUS is:

$$O\big(2n \times (2^q - 1)\big) = O(n2^q). \quad \text{Eq.1}$$

Giving the theoretic study of Hegland [11], the complication of Apriori algorithm is:

$$O(n * m^2), \quad \text{Eq.2}$$

Where m is the numbers of item in the databases.

Even though exponential form have in Eq.1, although polynomial form have in Eq.2, Eq.1 usually produces low value while comparing to Eq.2 for utmost existing transaction database. In truth, Eq.1 is exponential w.r.t the parameters q, i.e. the max numbers of itemset produced from the transaction database, size is not the problem (number of transaction in DB). In practicing, the q value is generally very low than the numbers of item in databases m. for example, the familiar case of hypermarket basket analysis, the numbers of product sale by a hypermarket can be numerous thousand while the average numbers of product buy by the individual customer exceeds a limited dozen.

Table 2.Theoretic Execution time complexity comparison of FIMUS and Apriori using Database.

Dataset	*n*	*m*	*q*	*FIMUS*	*Apriori*
BMS-WebView-1	*59602*	*497*	*2.5*	*14*	*247007*
BMS-WebView-2	*77512*	*3340*	*5*	*62*	*11155602*
Retail	*88162*	*16469*	*10*	*2046*	*271227971*
Connect	*100000*	*999*	*10*	*2046*	*998011*

In table 2 present the comparative between FIMUS and Apriori using the real time FIM dataset named BMS-WebView-1, BMS-WebView-2,Retail,Chess,Connect and Mushrooms are taken .All the dataset downloaded from SPMF Library[12] are seen in Table 2. The columns "FIMUS" and "Apriori", in specific, shown an approximation of the number of execution operation essential to build on the theoretic studies for the both algorithm present in this sections. The table reveal that for lesser instance, though the Apriori algorithms give superior result comparing to FIMUS in term of numbers of execution operation. But, for average and big instance, FIMUS visibly overtakes Apriori. Those results are complete by the Results and Discussion studies present in the next section. To conclude, FIMUS is additional scalability than the Apriori and have lesser computation value for database with average and huge numbers of item *m*.

IV. RESULTS AND DISCUSSION

The execution time implement of the Apriori and FP-Growth heuristic and FIMUS using the regular FIM dataset. Chess and Mushrooms dataset is implemented and it is a dense dataset, Chess have the number of transaction is 3196 and number of distinct items is 75 then the average transaction length 35, and for Mushrooms dataset have the number of transaction is 8124 and number of distinct items is 119 then the average transaction length 23. The Apriori beats FIMUS for dense data. In table 3, 4 the output approves that our method is superior to Apriori when dealing with dense data. The Execution time presentation shown in fig.3 and fig.5 the FIMUS and Apriori methods using the chess and Mushrooms data which is dense with various minimum support. The various minimum support from 40% to 90%, the execution time of FIMUS is remains stable while the Apriori is very highly increases. In fig4 and fig6, the various minimum support from 40% to 90%, the memory of FIMUS consume less memory than Apriori.

Table3. Execution time and memory on Chess dataset

Execution time(ms) and Maximum Memory(mb) on Chess Dataset				
Min_Sup	*FIMUS*		*APRIORI*	
	ET	*MM*	*ET*	*MM*
0.9	*23*	*63*	*199*	*124*
0.8	*34*	*44*	*3989*	*146*
0.7	*39*	*40*	*12258*	*194*
0.6	*31*	*77*	*115047*	*189*
0.5	*27*	*57*	*119970*	*182*
0.4	*160*	*39*	*137967*	*176*

These outcomes approve that FIMUS is not complex to variation of the minimum support threshold. This is clarified by considering that FIMUS is a transactions based methodology, in where the numbers of generated candidate Itemset is immovable not matter the supports used in the input. Equally, the Apriori heuristics is an items based methodology, in which the numbers of generated candidates increase when the minimum supports is minimized.

Table4. Execution time and memory on Mushrooms dataset

Execution time(ms) and Maximum Memory(mb) on Mushrooms Dataset				
Min_Sup	*FIMUS*		*APRIORI*	
	ET	*MM*	*ET*	*MM*
0.9	24	133	50	145
0.8	33	109	65	129
0.7	41	85	98	98
0.6	52	68	120	85
0.5	63	70	148	82
0.4	90	68	187	98

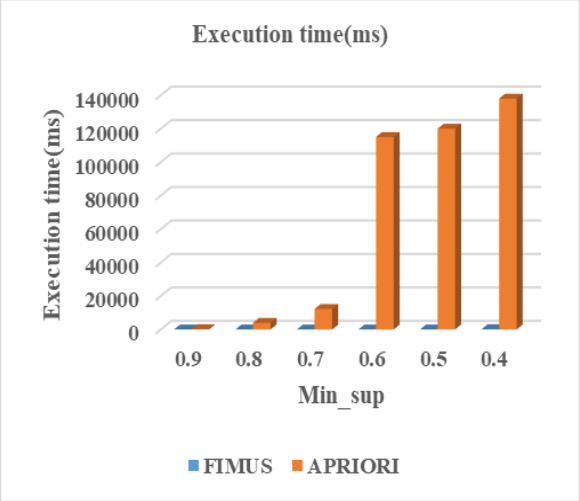


Fig.3 Execution time on chess data

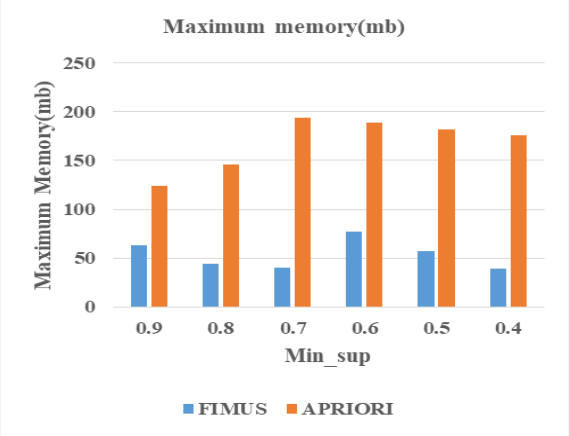


Fig.4 Maximum memory for chess data

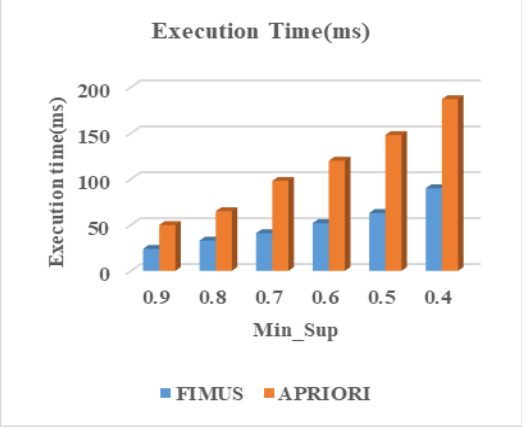


Fig.5 Execution time on Mushrooms data

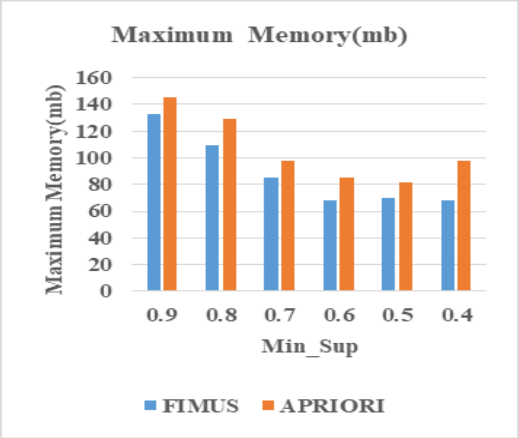


Fig.6 Maximum memory for Mushrooms data

V. CONCLUSION

The proposed FIMUS, a new intellectual Frequent Itemset mining algorithm. FIMUS extract Frequent Itemset with unique scan of a database. The candidates' itemset is first generated from every transactions and a hashtable is used to retain path of the fractional frequency of event of candidates' itemset though processing transaction. The both experimental and theoretical estimate tell that FIMUS outperform the Apriori heuristic for dense datasets. The scalability of FIMUS proven when various minimum support constraint. Feature work can extends FIMUS for solving big data problems, i.e. business intelligence for instance Internet of things, web event logs etc…mining data.

REFERENCES

1. R.Agrawal, T.Imielinski, and A.Swami" Mining association rules between sets of items in large databases". In: ACM SIGMOD Record, volume 22, June 1993.
2. J.Han, J.Pei, and Y.Yin: "Mining frequent patterns without candidate generation". In: ACM SIGMOD Record, volume 29, May 2000.
3. B.Liu, W.Hsu and Y.Ma: "Mining association rules with multiple minimum supports", Proc. ACM, SIGKDD conf. Sandiego, CA, USA, pp.337-341, 1999.
4. S.Brin, R.Motwani, J.D.Ullman, S. Tsur: "Dynamic itemset counting and implication rules for market basket data". In: ACM SIGMOD Record, vol. 26, no. 2, pp. 255–264. ACM, June 1997.
5. X. Wu, C. Zhang, and S. Zhang: "Efficient mining of both positive and negative association rules", ACM Tran. On Information Systems, 22(3), pp.381- 405, 2004.
6. Ling Zhou, Stephen Yau: "Efficient association rule mining among both frequent and infrequent items" Jou. Computers and Mathematics with Applications 54(6), pp.737–749. 2007.
7. M. Hegland: "The apriori algorithm tutorial". Math. Comput. Imaging Sci. Inf. Process. 11, 209–262 (2005).
8. M.J. Zaki, S. Parthasarathy, M. Ogihara, W. Li: "New algorithms for fast discovery of association rules". In: Third International Conference Knowledge Discovery and Data Mining (1997).
9. K.Amphawan, P. Lenca, A. Surarerks: "Efficient mining top-k regular-frequent itemset using compressed tidsets". In: L.Cao, J.Z. Huang, J. Bailey, Y.S. Koh, J. Luo (eds.) PAKDD 2011. LNCS (LNAI), vol. 7104, pp. 124–135. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28320-8 11
10. L. Cerf, J. Besson, C. Robardet, and J.F. Boulicaut: "Closed patterns meet n-array relations". ACM Trans. Knowl. Discover. Data (TKDD) 3(1), 3 (2009).
11. M. Hegland: The apriori algorithm tutorial. Math. Comput. Imaging Sci. Inf. Process. 11, 209–262 (2005).

12. P.Fournier-Viger, A.Gomariz, T. Gueniche, A. Soltani, C.W. Wu & V.S. Tseng (2014). "SPMF: a Java open-source pattern mining library." http://www.philippe-fournier-viger.com/spmf/.
13. C.K.S. Leung, M.A.F. Mateo, D.A. Brajczuk: "A tree-based approach for frequent pattern mining from uncertain data". In: T.Washio, E.Suzuki, K.M.Ting, A.Inokuchi, (eds.) PAKDD 2008. LNCS (LNAI), vol. 5012, pp. 653–661. Springer, Heidelberg (2008). Doi: 10.1007/978-3-540-68125-0 61.
14. G.Grahne, J. Zhu: "Fast algorithms for frequent itemset mining using FP-trees". IEEE Trans. Knowl. Data Eng. 17(10), 1347–1362 (2005).
15. C. Borgelt: "Frequent itemset mining". Wiley Interdisc. Rev.: Data Min. Knowl. Discover. 2(6), 437–456 (2012).
16. Y.Djenouri, H. Drias, and Z. Habbas:" Bees swarm optimization using multiple strategies for association rule mining". Inter. J. Bio-Inspired Comput. 6(4), 239–249 (2014).
17. J.M.Luna, M. Pechenizkiy, S.Ventura: "Mining exceptional relationships with Grammar-guided genetic programming". Knowl. Inf. Syst. 47(3), 571–594 (2016).
18. Y.Gheraibia, A.Moussaoui, Y.Djenouri, S. Kabir, and P.Y Yin: Penguins search optimization algorithm for association rules mining. CIT J. Comput. Inf. Technol. 24(2), 165–179 (2016).

AUTHORS PROFILE

A. Subashini, completed her M.Phil degree in the year 2008 at Annamalai University, at present doing her PhD degree in Anamalai University. She has fourteen years of teaching experience. Presently she is working as Assistant Professor in the Department of Computer Application at Government Arts College, C.Mutlur, Chidambaram. Her research interests are Neural Networks and Data Mining.

Dr. M. Karthikeyan, received the PhD degree from Annamalai University. Presently he is working as Assistant Professor in the Division of Computer & Information Science, Faculty of Science, Annamalai University. He published ten research papers in International journals and eight research papers in national journals. He has nineteen years of teaching experience and five years of research experience. His area of specialization includes neural networks & Fuzzy systems, Data Mining and Digital Image processing.