iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 5 **Issue:** XI **Month of publication:** November 2017

DOI: http://doi.org/10.22214/ijraset.2017.11058

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887
Volume 5 Issue XI November 2017- Available at www.ijraset.com

NCFP-tree: A Non-Recursive Approach to CFP-tree using Single Conditional Database

R.Prabamanieswari[1], D.S.Mahendran[2], T.C. Raja Kumar[3]
[1]*Associate Professor, Department of Computer Science, Govindammal Aditanar College for Women, Tiruchendur, India*
[2] *Associate Professor, Department of Computer Science, Aditanar College of Arts & Science, Tiruchendur, India*
[3]*Associate Professor, Department of Computer Science, St. Xavier's College, Tirunelveli, India*

Abstract: ***The CFP-tree is one of the efficient FP-tree based mining algorithms to find frequent itemsets. It stores all frequent item sets in compact form. It is more disk-friendly than FP-tree. But, it creates multiple conditional databases during tree construction and it takes more I/O and CPU consumptions. It also has redundancy in creating the conditional databases. In order to avoid these problems, the non-recursive algorithm NCFPGEN for creating a NCFP-tree is proposed in this paper. The NCFP-tree is similar to CFP-tree but, it is created in non-recursive manner. The proposed algorithm creates an extended conditional database instead of creating multiple conditional databases in CFP-tree. The experimental results show that our method outperforms the existing method such as CFP-tree in both memory consumption and execution time aspects. This proposed NCFP-tree can be utilized in any frequent itemset mining based algorithms such as association rule mining, classification and representative patterns set generation.***
Keywords: ***FP- tree like structures, frequent itemset, representative patterns set.***

I. INTRODUCTION

Knowledge discovery, whose objective is to obtain useful knowledge from data stored in large recognized as a basic necessity in many areas, especially those related to business. Since data represent a certain real-world domain, patterns that hold in data show us interesting relations that can be used to improve our understanding of that domain. Data mining is the step in the knowledge discovery process that attempts to discover novel and meaningful patterns in data. It is the process of extracting previously unknown potentially useful hidden predictive information from large amounts of data. Studies of Frequent Itemset (or Pattern) Mining is acknowledged in data mining field because of its broad applications in mining association rules, correlations and graph pattern constraint based on frequent patterns, sequential patterns and many other data mining tasks. Efficient algorithms for mining frequent itemsets are crucial for mining association rules as well as for many other data mining tasks. Different methods introduced by different researchers generated the frequent itemsets by using candidate generation process [1] as well as without candidate generation process [9] within which further division is based upon traversal such as depth-first traversal vs. breadth-first traversal and underlying data structures such as tree structure vs. other data structure. Many data mining problems are best represented with the help of non-linear data structures. The use of non-linear data structures in many interesting problems has spurred the interest of data mining researchers in the development of efficient and scalable data mining techniques for these special data structures. Trees, in particular, have recently attracted the attention of the research community, in part because they are particularly amenable to efficient pattern mining techniques. Identifying frequent patterns in a database of trees is an important task in solving many tree mining problems.

Many algorithms and techniques based on tree [9] are posed for enumerating itemsets from transactional databases. Let the transactional database D= $\{t_1, t_2, \ldots, t_n\}$, where t_j is a transaction containing a set of items, $j \in [1,n]$. Let $I = I_1, I_2 \ldots, I_m$ be a set of m distinct attributes and t be transaction that contains a set of items such that $T \subseteq I$. Each subset of I is called an itemset. If an itemset contains k items, then the itemset is called a k-itemset. The support of itemset X in database D is defined as the percentage of transactions in D containing X, that is, support $t_D(X) = \{t \mid t \in D \text{ and } X \subseteq t\}/D$. If the support of a pattern X is larger than a user specified threshold min-sup (min-sup $\in (0, 1]$), then X is called a frequent pattern. Given a transaction database D and a minimum support threshold min-sup, the task of mining frequent pattterns is to find all the frequent patterns in D with respect to min-sup.

Several algorithms such as COFI-tree [8], CT-PRO [15], CFP-tree [11] etc have been proposed based on FP-tree to find the frequent patterns. Among these, the CFP-tree structure is more preferable because it stores all frequent patterns in compact form. However, CFP-tree creates separate conditional databases for each itemsets which are to be processed and performs push-right step until all the conditional databases are processed. It places the last accessed non-empty conditional database onto disk to release some space

©IJRASET (UGC Approved Journal): All Rights are Reserved

in the memory if main memory is not large enough to hold all the conditional databases during the creation of conditional databases. Then, during the processing of conditional databases, the next conditional database is placed into the memory.These take more memory consumption and more number of memory load and unload processes. In this paper, the algorithm NCFPGEN (Non-recursive Condensed Frequent Pattern Generation) is proposed to reduce the problems of CFP. This proposed algorithm is similar to Condensed Frequent Pattern tree (CFP-tree) algorithm but it creates a single extended conditional database instead of creating multiple conditional databases. It does not always create a conditional database when a new itemset is considered for processing. It creates a single extended conditional database initially and its size is reduced each time. The conducted experiment shows that the proposed algorithm is more efficient than the existing CFP-tree approach.

The rest of the paper is organized as follows: Section II presents the related work. Section III describes the construction of NCFP-tree with example. The experimental results are shown in section IV. Finally, section V concludes the paper.

II. RELATED WORK

Many frequent pattern mining algorithms have been proposed in the literature. These algorithms are usually derived from one of the following two frequent pattern mining algorithms: Apriori [1] or FP-Growth [9]. The Apriori algorithm adopts candidates' generations-and-testing methodology to produce the frequent itemsets. In the case of long itemsets, the Apriori approach suffers from the lack of the scalability, due to the exponential increasing of the algorithm's complexity. The FP-Growth method explores some compressed data structure such as FP-tree. The FP-tree is a compact representation of all relevant frequency information in a database. The compression is achieved by building the tree in such a way that overlapping itemsets share prefixes of the corresponding branches. The FP-tree has a header table associated with it. Single items and their counts are stored in the header table in decreasing order of their frequency. The entry for an item also contains the head of a list that links all the corresponding nodes of the FP-tree. Apriori [1] and its variants [2] which need several database scans but, the FP-Growth method [9] needs only two database scans when mining all frequent itemsets. The first scan counts the number of occurrences of each item. The second scan constructs the initial FP-tree which contains all the frequency information of the original dataset. Mining the database then becomes mining the FP-tree. The FP-tree can be searched by following the depth-first strategy.The method FP-Growth is about an order of magnitude faster than the Apriori.

Several algorithms implicate the methodology of the FP-Growth algorithm. The papers [4]-[6] and [10] adapt the similar approach of [9] for mining frequent itemsets from the transactional database. But, these algorithms are more efficient than FP-Growth. Hajj and Zarane [8] present the Co-Occurrence Frequent Item tree (or COFI-tree for short) for mining frequent patterns. The presented algorithm is done in two phases. CT-PRO [15] is also a variation of classic FP-tree algorithm [9]. It is based upon the compact tree structure [16] for efficiently mining frequent patterns. It traverses the tree in a bottom up fashion. It constructs the compact FP-tree through mapping into index and then mine frequent itemsets according to projections index separately. In [5], the algorithm FPgrowth* is introduced. It uses simple additional data structure array. The array $A_{\varnothing}$ is constructed during the second scan for constructing $T_{\varnothing}$, for each transaction. It keeps the counts of all pairs of frequent items. From a conditional FP-tree T_x, when we construct a new conditional FP-tree for $X \cup \{i\}$, for an item i, a new array $A_{x \cup \{i\}}$ is calculated i.e., during the construction of the new FP-tree $T_{x \cup \{i\}}$, the array $A_{x \cup \{i\}}$ is filled. The construction of arrays and FP-trees continues until the FPgGrowth* method terminates. The array technique works very well especially when the dataset is sparse. However, when a dataset is dense, accumulating counts in the associated array may not be a good idea.

It has been observed that the complete set of frequent patterns often contains a lot of redundancy i.e.) many frequent patterns have similar items and supporting transactions. To overcome this problem, several approaches have been made to construct a concise representation of frequent itemsets. Two major approaches have been developed in this direction: lossless compression and lossy approximation. To construct a concise representation based on lossless compression, methods such as closed frequent patterns [12] and a non-derivable itemsets [3] are used. The frequent closed patterns give the less number of frequent representative patterns and all the frequent itemsets can be derived from them. Most applications will not need precise support information for frequent patterns: a good approximation for the support count could be more than adequate. The ideas of approximating frequent patterns from representative sets have been discussed in [13]. The good approximation algorithms such as RPglobal and RPlocal [14] need to perform substantial coverage checking that checks whether an itemset can be covered by another one in order to find representative patterns. They are very time-consuming and space-consuming. To improve the performance, RPglobal and RPlocal have to use some FP-tree like structures to index frequent itemsets and representative itemsets to reduce the number and the cost of coverage checking. The approximation algorithm MinRPset [7] utilizes several techniques to reduce the running time and the memory usage.

©IJRASET (UGC Approved Journal): All Rights are Reserved

In particular, it uses a tree structure called CFP-tree [11] to store frequent patterns compactly to find subsets when determining representative patterns set.

The existing concise representation approaches such as closed frequent patterns [12] and a non-derivable itemsets [3] reduce the result size but they increase the cost needed to drive all frequent itemsets from these concise representations regardless of the physical storage of the concise representation. Studies of Frequent Itemset (or Pattern) Mining is acknowledged in data mining field because of its broad applications. The CFP-tree algorithm plays an important role for finding frequent itemsets. It takes lesser effort for driving all frequent itemsets. And also, it can be used by other mining approaches. However, the CFP-tree algorithm suffers from memory load and unload problem. To reduce this problem, the non-recursive algorithm NCFPGEN for creating a NCFP-tree (Non-Recursive CFP-tree) is proposed in this paper. The proposed approach always takes less memory space and less execution time comparing to CFP-tree.

III.CONSTRUCTION OF NCFP-TREE

The construction of proposed NCFP-tree (Non-Recursive CFP-tree) is similar to CFP-tree structure but, it differs in processing the conditional database. The main difference is that a new single extended conditional database is created initially instead of creating multiple conditional databases. In CFP-tree, a new conditional database is created for each itemset every time by applying push-right step. Here, the extended conditional database is utilized in an efficient manner by keeping the transactions start with the itemsets which are not yet processed. It does not follow the recursive approach. The overview of the proposed approach is given below:

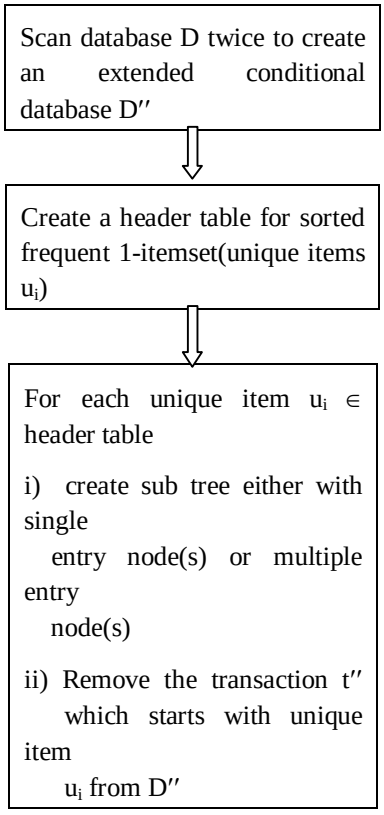


Fig 1. Overview of NCFP-tree approach

A. Framework

Given transactional database D and a minimum support threshold, the NCFP-tree construction algorithm (NCFPGEN) scans the database D first to find all frequent 1-itemset and sort into ascending frequency order denoted as $F=\{i_1,i_2\ldots\ldots i_n\}$. In the second database scan, a conditional database D′ and an extended conditional database D″ are constructed. The conditional databases D′ are constructed as follows: For each transaction t in D, infrequent items are removed and the remaining items are sorted according to their orders in F. The extended conditional database D″is constructed for each transaction t′ in D′. It is constructed as follows: Given

©IJRASET (UGC Approved Journal): All Rights are Reserved

a transaction t′ in D′, a new transaction t″ which starts with i_j is created for each item i_j in t′ except for i_1 & i_n. The extended conditional database D″contains the complete information for mining frequent itemsets. The mining is performed on the extended conditional database D″ only. There is no need to access the original database D and the conditional database D′. We use a header table to maintain the set of frequent 1-itemset in a database. The frequent 1-itemset is sorted into ascending frequency order in the header table. The extended conditional database D″ is processed according to the order of the frequent items in header table. The creation of NCFP-tree starts from left to right for each frequent item in the header table. It creates sub tree for each frequent item u_i in the header table. The algorithm is given below:

1) *Algorithm:* NCFPGEN
2) *Input:* D is the database min-supp is the minimum support threshold
3) *Description:*
a) Find frequent 1-itemset based on min-supp from the original database D & sort into ascending frequency order as F={i_1,i_2……i_n}
b) Create a conditional database D′for each transaction t ∈ D create a new transaction t′ with the frequent items according to the order in F & remove infrequent items
c) Create an extended conditional database D″ //creation of extended conditional database D″ by extending conditional database D′for each transaction t′ ∈ conditional database D′ do
d) Create a header table for sorted frequent 1-itemset (unique items u_i)
e) Create NCFP-tree with sub trees correspond to each unique item u_i

```
for each unique item u_i ∈ header table do
{       //sub tree construction
            process-itemset =u_i
            while (process-itemset ! = null)
            {
                locate the transactions t″ which start with process-itemset from D″
                find support count for process-itemset
                for each remaining item i_j∈ t″ except process-itemset
                    Find support count for i_j
                for each remaining item i_j∈ t″ except process-itemset
                {
                    if(supp-count( i_j )/supp-count(process-itemset)>=min-supp)
                        if (supp-count( i_j ) = supp-count(process-itemset))
                        //combine i_j with process-itemset & create a node with single entry
                        process-itemset = process-itemset ∪ i_j
                        store process-itemset & support count
                else
                        //create a node with multiple entries & repeat for all entries
                           strore  i_j  & support count value in each entry
                           process-itemset = process-itemset ∪ mul-entry i_j
                }
            }//while
        Remove the transactions t″ which start with u_i from D″
}
```

f) End

B. Example

Consider the following Transactional Database. It has five transactions, that is |D|=5. Assume min-sup=40%. The frequent 1-itemset is determined and is sorted. It is denoted as F={i_5:2, i_1:3, i_2:3, i_3:3, i_4:5}. The Conditional Database D′ and Extended Conditional Database D″ are given in Table II and Table III respectively.

©IJRASET (UGC Approved Journal): All Rights are Reserved

TABLE I
TRANSACTIONAL DATABASE DATABASE

Trans_ID	List of Items
T_1	i_1,i_2,i_3,i_4
T_2	i_2,i_3,i_4,i_5
T_3	i_1,i_2,i_4
T_4	i_3,i_4,i_5
T_5	i_1,i_4

TABLE II
CONDITIONAL DATABASE

Trans_ID	List of Items
T_1	i_1,i_2,i_3,i_4
T_2	i_5,i_2,i_3,i_4
T_3	i_1,i_2,i_4
T_4	i_5i_3,i_4
T_5	i_1,i_4

TABLE III.
EXTENDED CONDITIONAL

Trans_ID	List of Items
T_1	i_1,i_2,i_3,i_4
T_2	i_5,i_2,i_3,i_4
T_3	i_1,i_2,i_4
T_4	i_5,i_3,i_4
T_5	i_1,i_4
T_6	i_2,i_3,i_4 (T_1)
T_7	i_3,i_4
T_8	i_2,i_3,i_4 (T_2)
T_9	i_3,i_4
T_{10}	i_2,i_4 (T_3)
T_{11}	i_3,i_4 (T_4)

TABLE IV
FREQUENT ITEMSETS (COMPACT FORM)

Frequent Itemsets(Compact Form) (min_sup = 40%)
$i_5i_3i_4$:2
$i_1i_2i_4$:2 i_1i_4 :3
i_2 i_3i_4 :2 i_2i_4 :3
i_3i_4 :3
i_4 :5

TABLE V
ALL POSSIBLE ITEMSETS

All Patterns (min_sup = 40%)
i_5:2 i_1:3, i_2:3, i_3:3, i_4:5,
i_1 i_2:2, i_1i_4:3, i_2 i_3:2, i_2i_4:3 i_3 i_4:3, i_3 i_5:2, i_4 i_5:2
i_1i_2 i_4:2, i_2i_3 i_4:2, i_3i_4 i_5:2

Table V shows more number of frequent itemsets based on min-supp. But, our approach gives reduced number of frequent itemsets which is shown in Table IV.

The NCFP-tree constructed for all unique item u_i in the header is given in Fig. 2. It shows all frequent items which correspond to each unique item u_i as a separate sub tree in compact form. We can drive all the possible frequent itemsets of Table V from each sub tree easily. Therefore, this tree can be used in any mining approaches in lesser effort.

©IJRASET (UGC Approved Journal): All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887
Volume 5 Issue XI November 2017- Available at www.ijraset.com

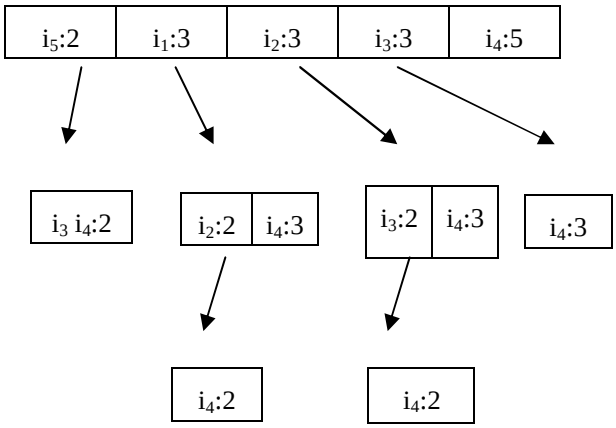


Fig2. NCFP-tree

IV. EXPERIMENTAL RESULTS

The experiments are carried out on the computer with the configuration such as Intel(R) Core(TM) i3CPU, 3 GB RAM, 2.53 GHz Speed and Windows 7 Operating System. The CFP-tree and NCFP-tree approaches are implemented in Java. The experiments are evaluated on three datasets. The datasets are mushroom, retail and T10I4D100K. The mushroom and retail are real datasets. They are relatively dense. The dataset T10I4D100K is synthetic and is quite sparse. The mushroom dataset contains the characteristics of various species of mushrooms. It has 119 items and 8124 transactions. The minimum, maximum and average length of its transaction is 23.The retail dataset contains the retail market basket data from an anonymous Belgian retail store. It has 16,470 items and 88,162 transactions. The maximum length of its transaction is 77 and the average length of its transaction is10.31. Both are obtained from the UCI repository of machine learning databases. The synthetic dataset T10I4D100K is obtained from IBM dataset generator, which consists of 100,000 transactions with 1000 items and an average length of 10 items.

The two algorithms are tested on the mushroom dataset with a support level of 10% to 90% in increments of 10%.They are also tested on the retail dataset and T10I4D100K dataset with a support level of 0.01 to 0.05 and 0.03 to 0.07 respectively in increments of 0.01.

A. Memory Consumption

The proposed NCFP-tree consumes lesser memory space than CFP-tree. For mushroom dataset, when min-supp is high, the difference of memory space consumption between CFP-tree and NCFP-tree approaches is small. When min-supp is low, difference of memory space consumption is high. Fig. 3 (a) shows the memory space consumption of the two approaches when running them on mushroom dataset. Fig. 3 (b) shows the memory space used by the algorithms when retail dataset is used and Fig. 3 (c) shows the memory size used by the algorithms when T10I4D100K dataset is used. From Fig. 3, we can see that the memory space consumption of NCFP-tree is always less comparing to CFP-tree.

©IJRASET (UGC Approved Journal): All Rights are Reserved

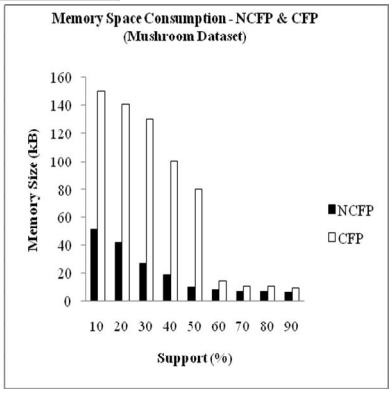


Fig. 3 (a)

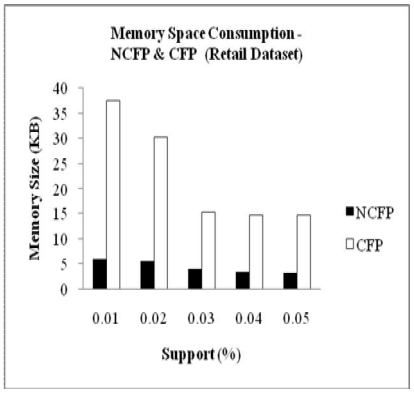


Fig. 3 (b)

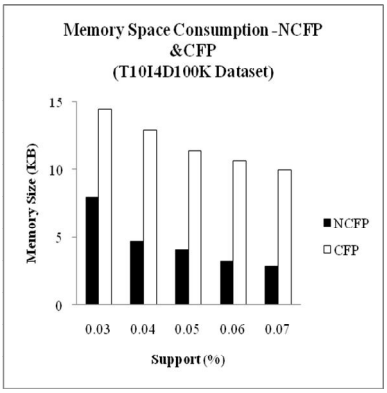


Fig. 3 (c)

Fig. 3 (a) – 3(c). Memory Space Consumption of NCFP-tree and CFP-tree

B. Running Time

The proposed NCFP-tree is always better than CFP-tree. G.Liu et al [11] discussed that the time of constructing a CFP-tree for storing all frequent itemsets includes both CPU time and I/O time. In our proposed approach, a single extended conditional database is used. Therefore, it does not have memory load and unload problem. The experiment includes only CPU time for constructing a NCFP-tree. Fig. 4 (a) - 4(c) shows the running time of NCFP-tree and CFP-tree.

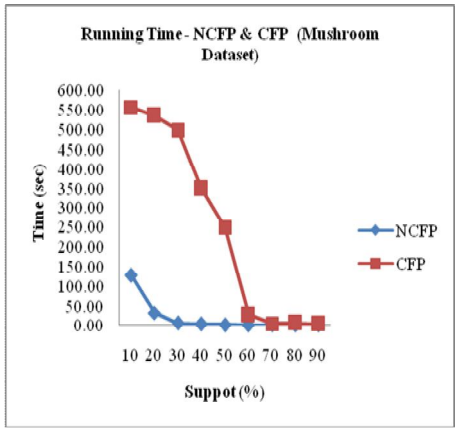


Fig. 4 (a)

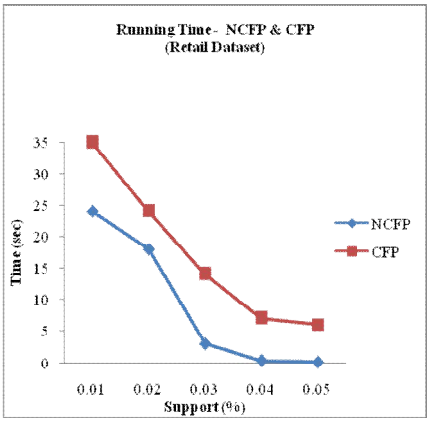


Fig. 4 (b)

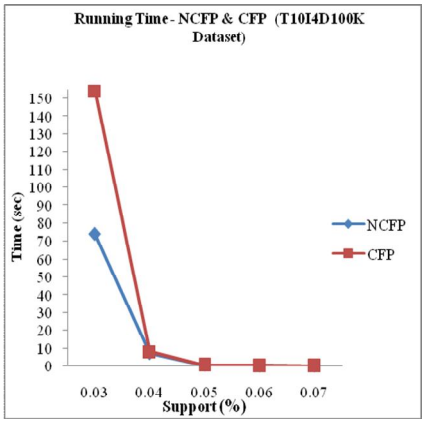


Fig. 4 (c)

Fig.4 (a) - 4 (c). Running Time of NCFP-tree and CFP-tree

V. CONCLUSION

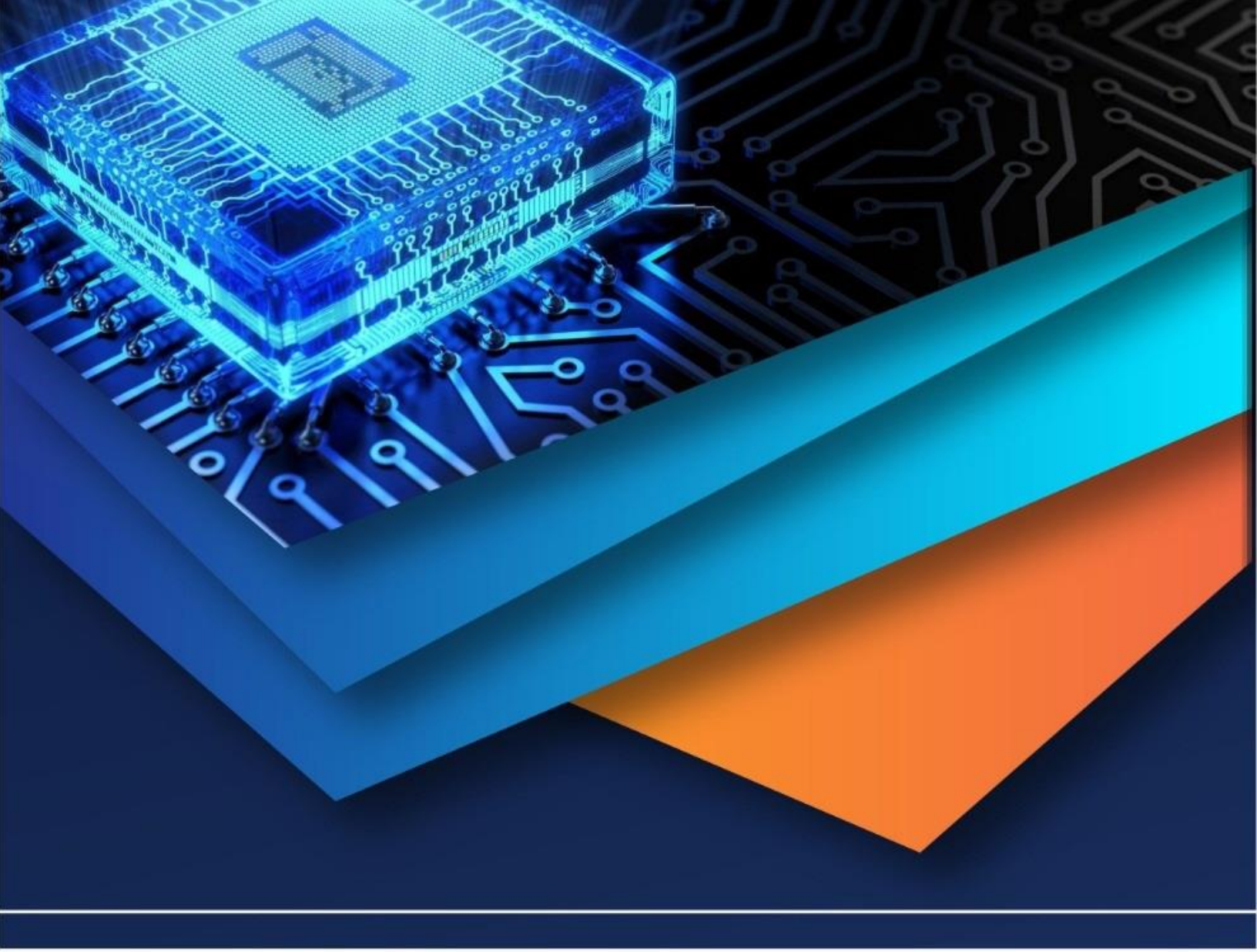
The major challenge found in frequent pattern mining is a large number of resultant patterns. Recent studies on frequent itemset mining algorithms resulted in significant performance improvements.The FP-tree like structure such as CFP-tree reduces the number of frequent patterns by storing all frequent patterns in compact form. The CFP-tree also supports for retrieving frequent patterns to find the concise representation such as closed itemset, representative patterns set etc. In this paper, we have proposed NCFP-tree which is similar to CFP-tree algorithm but, it creates a single extended conditional database instead of creating multiple conditional databases. Here, the extended conditional database is utilized in an efficient manner by keeping the transactions start with the itemsets which are not yet processed. Our approach does not follow the recursive and push-right techniques. The proposed tree structure NCFP-tree also provides all features similar to CFP-tree but, it always takes less memory space and less execution time comparing to CFP-tree.

REFERENCES

©IJRASET (UGC Approved Journal): All Rights are Reserved

[1]. R. Agrawal, T. Imielinski and A.N. Swami, "Mining association rules between sets of items in large databases,"in Proc. SIGMOD, Washington, DC, USA 1993, pp. 207–216.
[2]. Borgelt, "Efficient implementations of apriori and éclat," In: Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03), 2003, Volume 90 of CEUR Workshop Proceedings, Melbourne, Florida, USA.
[3]. T. Calders and B. Goethals, "Mining all non-derivable frequent itemsets," In Proc. of 2002 European Conf. On Principles of Data Mining and Knowledge Discovery (PKDD'02), 2002, pp 74–85.
[4]. J.Gao,"Realization of new Association Rule Mining Algorithm," Int.Conf. On Computational Intelligence and Security, IEEE 2007.
[5]. G. Grahne and J. Zhu, "Efficiently using prefix-trees in mining frequent itemsets," in: Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, 2003.
[6]. G.Grahne and J.Zhu, "Fast Algorithm for frequent Itemset Mining Using FP-trees," IEEE Transactions on Knowledge and Data Engineer, vol. 17, no. 10, 2005.
[7]. Guimei Liu, Haojun Zhang and Limsoon Wong, "A Flexible Approach to Finding Representative Pattern Sets," IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 7, pp 1562-1574, 2014.
[8]. M.EI-Hajj and O.R.Zarane, "COFI-tree Mining: A new Approach to Pattern Growth with Reduced Candidancy Generation," Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, 19 December 2003, Melbourne, Florida, USA, CEUR Workshop Proceedings, vol. 90.
[9]. Han, J. Pei and Y. Yin, "Mining frequent patterns without candidate generation," In Proceedings of ACM SIGMOD'00, 2000, pp 1–12.
[10]. G. Liu, H. Lu, J. X. Yu, W.Wang and X.Xiao, "AFOPT: An Efficient implementation of Pattern Growth Approach," In Proc.IEEE ICDM'03 Workshop FIMI'03, 2003.
[11]. G. Liu, H. Lu, and J. X. Yu, "CFP-tree: A compact disk-based structure for storing and querying frequent itemsets," Inf. Syst., vol. 32, no. 2, pp. 295–319, 2007.
[12]. N. Pasquier, Y. Bastide, R. Taouil and L.Lakhal, "Efficient Mining of Association Rules using Closed Itemset Lattices," Information Systems, vol 24, no 1, pp 25-46,1999.
[13]. R.Prabamanieswari, D.S.Mahendran and T.C. Raja Kumar, "A Survey on Concise and Lossless Representation of Frequent Pattern Sets," International Journal of Advanced Research in Computer and Communication Engineering, vol. 4, issue 9, 2015.
[14]. Xin, J. Han, X. Yan, and H. Cheng, "Mining compressed frequent-pattern sets," in Proc. 31st Int. Conf. VLDB, Trondheim, Norway, 2005, pp. 709–720.
[15]. Y.G. Sucahyo and R.P.Gopalan, "CT-PRO: A Bottom up Non Recursive Frequent Itemset Mining Algorithm Using Compressed FP-tree Data structure," in Proc Paper presented at IEEE ICDM Workshop on Frequent Itemset Mining Implementation (FIMI), 2004, Brighton UK.
[16]. Y.G. Sucahyo and R.P.Gopala, "High Performance Frequent Pattern Extraction using Compressed FP trees," Proceedings of SIAM International Workshop on High Performance and Distributed Mining (HPDM), 2004, Orlando, USA.

©IJRASET (UGC Approved Journal): All Rights are Reserved



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)