International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/CSEIT206537

Mining Frequent Pattern by Titanic and FP-Tree algorithms

Youssef FAKIR, Rachid El AYACHI, Mohamed FAKIR
Laboratory of Information Processing and Decision Support, Faculty of Sciences and Technics, Sultan Moulay Slimane Unversity, Morocco

Article Info
Volume 6, Issue 5
Page Number: 208-215
Publication Issue :
September-October-2020

Article History
Accepted : 05 Oct 2020
Published : 12 Oct 2020

ABSTRACT

Extraction of itemset frequent is an important theme in Datamining. Several algorithm have been developed based on Apriori algorithm during the last decades. This paper deals with the FP- tree and Titanic algorithms. FP-Tree is an improvement to the Apriori method witch generate frequents itemsets without generating candidate. The Titanic algorithm traverses the level search space by focusing on the determination of the minimum generators (or key Item sets). In addition, this paper studies the differences between these two algorithms and shows advantages and disadvantages of each one.

Keywords : Data mining, Association rules, FP-Tree, TITANIC algorithm

I. INTRODUCTION

Datamining is an important part of the computer science, it includes a lot of technologies and sciences and techniques, such as statistics, data base theory, data science and machine learning. When we speak about datamining, we speak about the whole process of discovering, uncovering and exploring patterns in large data sets like Big Data.

During the last decade several algorithm have been developed [1, 2,3,4,5,6,7,8, 9]. Most of them are based on Apriori algorithms. Apriori scan the database on several passes. When the data is large, it become difficult. In order to avoid repetition of the database, Han [6] et al have proposed a method called FP-Growth to resolve this problem. The FP-growth algorithm [10, 11, 12] thus provides a solution to the problem of searching for frequent patterns in a large transactional database. By storing all common elements of the transaction database in a compact structure, the need to repeatedly scan the database of transactions is eliminated. Moreover, by sorting the elements in the compact structure, one accelerates the search of the reasons. To avoid the limitation of the Apriori, Stumme has proposed the TITANIC algorithm for the discovery of frequent closed item sets [3]. The key idea is to minimize the step of calculating the Support Item sets. Thus, the algorithm traverses the level search space by focusing on the determination of the minimum generators (or key Item sets) of the different classes of the equivalence relation induced by the closure operator.

The organization of this paper is as follows. Section 2, describes the FP-Tree method. Section 3 deals with Titanic algorithm. Experimental results are presented in Section 4. Section 5 gives out the conclusions.

II. FP-TREE

Frequent Pattern Tree is made with the initial itemsets of the database. The purpose of the FP tree is to mine the most frequent pattern. Each node of the FP tree represents an item of the itemset. Two essential elements constitute the structure of an FP-tree [2]. Therefore, the two elements that make up this structure are:

- A structure in the form of a tree with a root labelled 'null'.
- An index (a table of frequent item pointers).

The tree is therefore composed, as indicated, of a root 'null' and a set of nodes prefixed by the represented element.

A node of the tree is composed by:

- The name of the item, the item represented by the node.
- The transaction count number where the path portion is located to this node.
- A link to the next node in the tree (node-link). This is an inter-node link to other occurrences of the same element (with the same item-name) in other transaction sequences. This value is null if there is no such node.

The Index contains the list of frequent items and points to the first occurrence of each item. Each entry in this table contains:

- The name of the element (item-name).
- The leading pointer of the sequence of nodes having this same item-name.

The Major steps to Mine FP-tree are:

- Construct conditional pattern base for each node in the FP-tree
- Construct conditional FP-tree from each conditional pattern-base .
- Recursively mine conditional FP-trees
- If the conditional FP-tree contains a single path, simply enumerate all the patterns.

The frequent pattern growth method find the frequent pattern without generating candidate. The construction of an FP-tree structure goes through 6 main steps. Steps 1 through 5 prepare the structure and insert the elements that should be there. The sixth step is to validate the information inserted in the previous steps:

- Calculate the minimum support
- Browse the transaction database to find the total sum of the different occurrences.
- Define the priority of the items, and then sort the items according to their priority.
- Creating the root node.
- Insert child nodes.
- Validation

Step 1: The first step is to scan the database to find the occurrences of the itemsets in the database. This step is the same as the first step of Apriori. The count of 1-itemsets in the database (Table 1) is called support count or frequency of 1-itemset.

Table 1: Transaction database

TID	Items
1	f,a,c,d,g,I,m,p
2	a,b,c,f,l,m,o
3	b,f,h,j,o
4	b,c,k,s,p
5	a,f,c,e,l,p,m,n

Suppose the minimum support is set to 50 in our case, to get a support of 50%, it must be calculated as follows:

Support minimum = (50/100 * 5) = 2.5. 50 being the minimum percentage requested, five being the total number of transactions in the database. If the value obtained is not complete, it will be rounded. (For example: ceiling (50/100 * 5) = 3. The result obtained is 3 in our case, constitutes the minimum support and consequently all the items of the database of transactions having a support lower than 3 minimum occurrences will be ignored (Table 2).

This step being completed let us go to the next step.

Step 2: In this step, we will go through the database of transactions to calculate the frequencies of the elements that are there [3]. Subsequently, once the different frequencies obtained, only the elements whose frequency is greater than the minimum support defined in step 1 (table 1) will be retained, the others will be ignored. In our case, the following table represents the selected items and their respective occurrence numbers.

The table obtained constitutes the table of headers (Headers Table) also called the table of the pointers. Now that we have determined our different items, we must prioritize them. This task is performed in the next stage of construction.

Table 2: Items with their frequencies.

Item	Frequency
f	4
a	3
c	4
b	3
m	3
p	3

Step 3: This step consists of ordering the different elements according to their weight. They are sorted according to their number of frequencies (Table 3). This sorting is done in descending order, the element with the highest number of occurrences is placed at the top, and the element with the least number of occurrences is placed at the bottom. This processing will be done for each of the transaction lines contained in the transaction database.

Table 3: items with their frequencies ordered.

TID	Items	Frequent items arranged
1	f,a,c,d,g,I,m,p	f,c,a,m,p
2	a,b,c,f,l,m,o	f,c,a,b,m
3	b,f,h,j,o	f,b
4	b,c,k,s,p	c,b,p
5	a,f,c,e,l,p,m,n	f,c,a,m,p

Step 4: From the result obtained in the previous step (Table 3), we begin the construction of the FP-tree structure. First, the 'Root' element of the tree is created. This root element will not contain any element. It will only contain links to its child elements. We begin by going through each element of the transaction (figure 1). Then for each element of the transaction we check the existence of a corresponding node, if it does not exist, the node is created, otherwise the number of occurrences is incremented. Then for each element created we will establish a link from the header table to the element inserted in the tree. At this point the tree is still empty, so the procedure to check the existence of a particular node will indicate that it does not exist and therefore it will have to be created. The first transaction is composed of the elements (f, c, a, m,p) sorted in descending order according to their weight. Since the element f is the first in the list, a corresponding node is inserted from the root element of the tree. Node f contains the count of 1 because this is the first time, we have inserted this element. A

link is established between the root element of the tree and the element f and another link is established from the header table. The same goes for the following elements (c, a, m, p). Thus, we obtain the structure illustrated by this figure below.

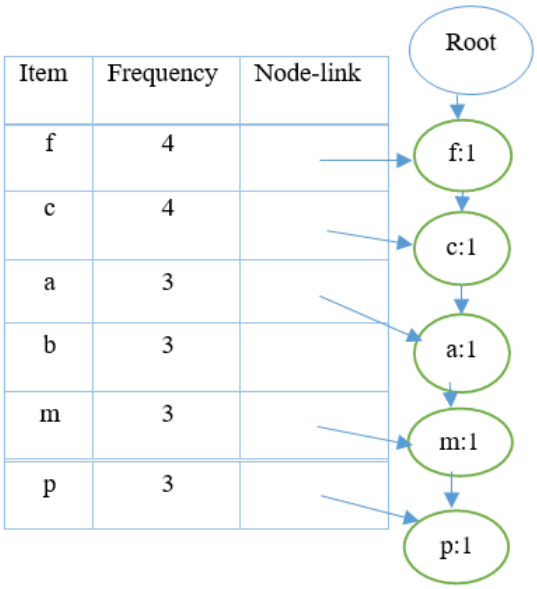


Figure 1: First creation of the FP-TREE

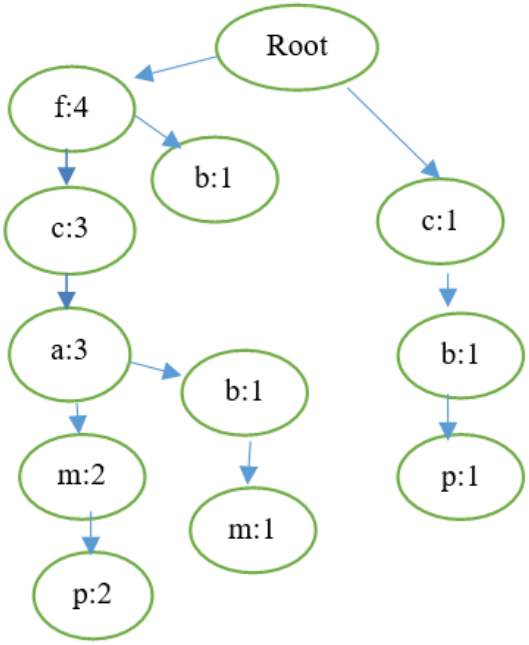


Figure 2: Final result of the FP-Tree

Step 5: The construction continues with the second transaction which is composed of elements (f, c, a, b, m) (figure 1). This time the tree contains elements and therefore for each element found its number of occurrences is incremented by 1. It is the same for the elements c, and a. We arrive at element b. There are no matching elements, so a new node is created from our current position, node a and a new link is created from a to b and then a link from the header table to the new inserted element. Regarding the remaining m element, since there is no corresponding node from our position (node b) a new node is created and initialized with a value of 1, then in addition to the link created from element b, a link is created from the already existing node m.

Step 6: This last step consists in validating the information of the tree (Figure 2). How to know if they are correct? The answer to this question is very simple; it is enough to compare the information obtained from the different nodes of the tree with the information of the header table. For that, it is necessary to count, and to add if necessary all the occurrences of an element in the tree and to compare the result obtained with that stored in the table of the headings (Table 1). Thus, after having counted all the elements of the structure, one obtains the following result: (f: 4, c: 4, a: 3, b: 3, m: 3, p: 3) which is in conformity with the table of headers. Table 4 shows the result of mining patterns by creating conditional patterns base.

Table 4: Conditional Pattern Bases

Item	Conditional Pattern Bases	Conditional FP-tree
p	{(fcam:2), (cb:1)}	{(c:3)}|p
m	{(fca:2), (fcab:1)}	{(f:3, c:3, a:3)}|m
b	{(fca:1), (f:1), (c:1)}	Empty
a	{(fc:3)}	{(f:3, c:3)}|a
c	{(f:3)}	{(f:3)}|c
f	Empty	Empty

III. TITANIC ALGORITHM

A. Pseudo code

The TITANIC algorithm has been proposed by Stumme for the discovery of frequent closed item sets. The key idea is to minimize the step of calculating the Support Item sets [5]. Thus, the algorithm traverses the level search space by focusing on the determination of the minimum generators (or key Item sets) of the different classes of the equivalence relation induced by the closure operator. The TITANIC algorithm instantiates the two steps of the generic algorithm: Pruning step and the construction step. The pseudo algorithm is as follows:

---

input: K : Extraction context, minsup
output: FC=$\cup_k FC_k$: Set of frequent closed itemsets

1: $FFC_0=\{\emptyset\}$ /*(FFC_K: Set of K-generators candidates. FC_K: Set of minimal frequent K-generators.)*/

2: ∅.support=| K |

3: FFC1={1-item sets}

4: for (k=1 ; FFC_k.gen 6= ∅ ;k + +) faire 5: FFCk= CALCUL-SUPPORT(FFCk)

6: FCk−1= closing-calculation (FFCk−1) {/* ∀ i ∈ FCk (i)00= i ∪{x ∈ A -{i} | support(i)= support(i ∪ {x}) } */}

7: FCk= Prune -Notfrequent-Nokey(FFCk)
{/* ∀ i ∈ FCk si support (i) < minsup ou ∃ j ∈ i | support(i)= support (i -{j})
*/}

8: FFCk+1=Genereate-Candidate(FFCk)

9: end for

10: return FC=∪kFCk

The particularity of this algorithm is that he considers the empty set ∅. In the Pruning step, the CALCUL-SUPPORT function is applied to each FFC generator k, this determining its support. Knowing that this calculation requires access to the extraction context. Note that each item is assigned an estimated support, which is equal to the minimum value of the media of the two joined items to obtain it. The infrequent k-generators are eliminated by calling the function ELAGUER-INFREQUENT-NOT- CLOSED. No-key k-generators, whose value of their estimated support is equal to the actual support value, are also eliminated. At this level, the algorithm TITANIC proposes to calculate the closures of the minimal generators which were retained during the previous iteration, those belonging to FCk¡1.

B. Illustration of Titanic

The tables illustrate the execution trace of the TITANIC algorithm on the extraction context K for minsup=2. During the initialization phase (Table 5), the 1-generators of the set FC1 are considered as key item sets is pruned by calling the ELAGUE-INFREQUENT-NOT-CLOSED function to obtain the FC1 set (Table 6).

The set of 2 FFC2 (Table 7) candidate generators is obtained by calling the GENERATE-CANDIDATE function, which performs a self-join on the set FC1. The CALCUL-SUPPORT function determines the support of each candidate of FCC2 . Non-frequent and non-key 2-generators are eliminated. For example, the frequent 2-generators AC and BE are eliminated, since their estimated support is equal to the value of their real support. During this iteration, the closures of the elements of the set FC1 are calculated. The function GENEREATCANDIDAT is applied to the set FC2 to obtain the set FFC3 composed of 3-generator candidates (Table 12).

The single item in this set is pruned by calling the function ELAGUE-INFREQUENT-NOT-OUT. Before ending, since FC3 is empty, the algorithm calculates the closures of the 2-generators retained in FC2 (Table 10). Table 14 illustrate the final result. each element of this set is composed of 4

field: the item (generator), Estimatif support (S-E), Real support (S-R) and the Key (Yes/No).

Table 5: Transaction database table

ID	Item
1	A,C,D
2	B,C,E
3	A,B,C,E
4	B,E
5	A,B,C,E

Itemset	S-E		Key
Ø	5		Yes

Table 6: FFC1

Item	S-E	S-R	key
A	5	3	yes
B	5	4	yes
C	5	4	yes
D	5	1	yes
E	5	4	yes

Table 7: FC1

Item	S-E	S-R	key
A	5	3	yes
B	5	4	yes
C	5	4	yes
E	5	4	yes

Table 8: FC0 Closing calculation

key	**closing**
Ø	Ø

Table 9: FFC2

Item	S-E	S-R	key
AB	3	2	yes
AC	3	3	No
AE	3	2	yes
BC	4	3	yes
BE	4	4	no
CE	4	3	yes

Table 10: FC2

Item	S-E	S-R	key
AB	3	2	yes
AE	3	2	yes
BC	4	3	yes
CE	4	3	yes

Table 11: FC1 Closing calculation

key	closing
A	AC
B	BE
C	C
E	BE

Table 12:FFC3

Item	S-E	S-R	key
ABE	3	3	No

Table 13:FC2 Closing calculation

key	closing
AB	ABCE
AE	ABCE
BC	BCE
CE	BCE

Table 14: Output

key	Frequent closed itemset
A	AC
B	BE
C	C
E	BE
AB	ABCE
AE	ABCE
BC	BCE
CE	BCE

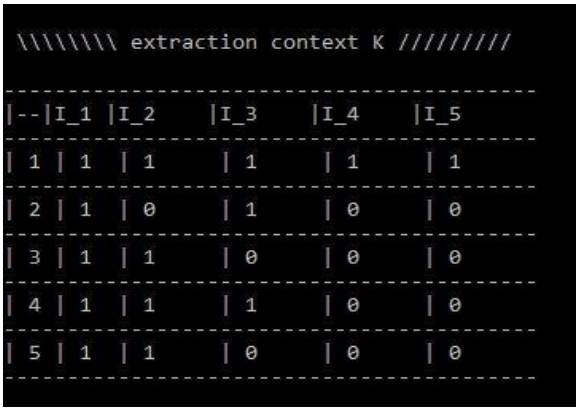
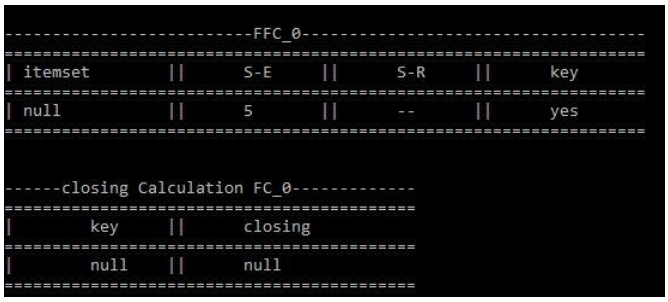


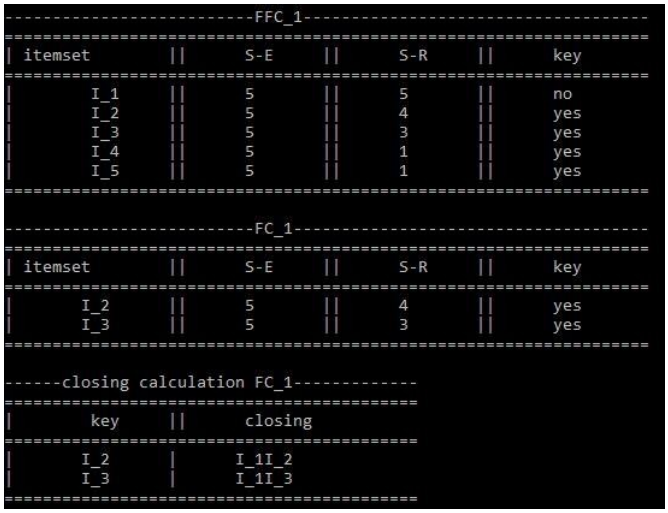
FIG.3: EXAMPLE OF TID

II. EXPERIMENTAL RESULTS

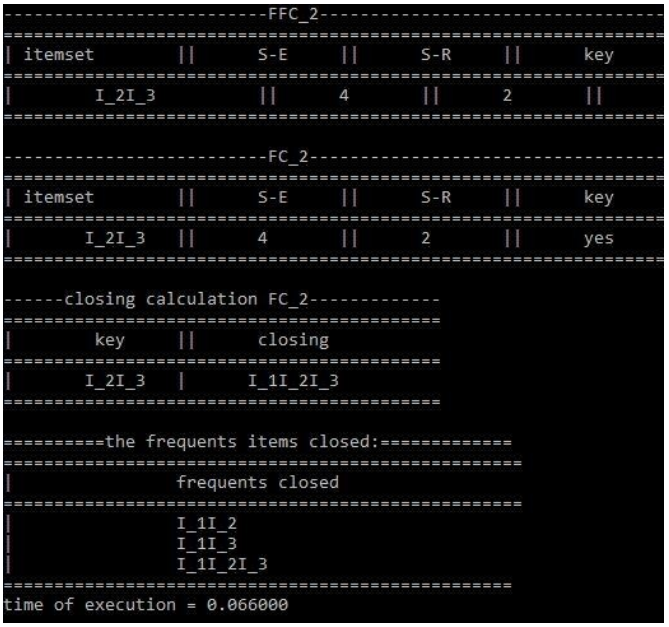
Figure 3 and Figure.4 show respectively the result obtained by applying Titanic algorithm and FP-Tree. The computing time is given seconds.



(a)



(b)



(c)

Fig.3: Execution of Titanic algorithm

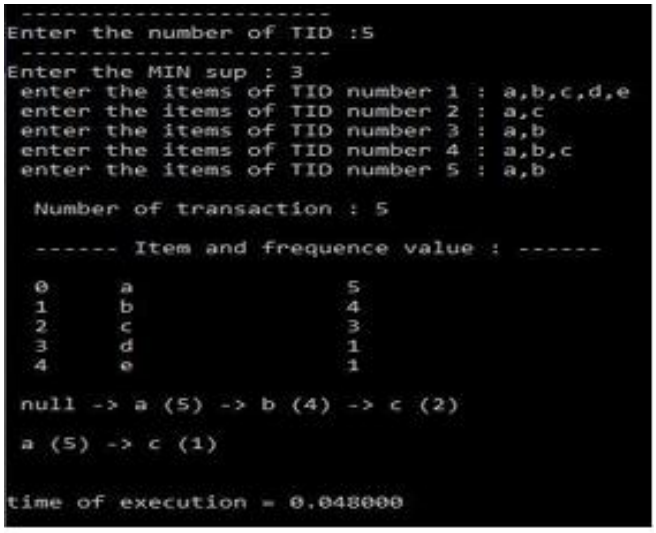


FIG.4: RESULT OBTAINED BY FP-TREE

IV. CONCLUSION

In this paper we described an implementation of a pattern growth based on frequent itemset mining algorithm called FP-TREE on the first hand, which is useful in cases when we have to deal with a large dataset and make a lot of passes and scans. As well as, FP-tree is the first of the dataminig algorithms that success in mining the frequent item sets. On the other hand, we worked on TITANIC algorithm that the key idea is to minimize the steps of

calculating the support item sets and to be able to traverse the level search space by focusing on the determination of the minimum generators (or key Item sets) of the different classes of the equivalence relation induced by the closure operator. In addition, for comparing these two algorithms, we have noticed that FP-TREE is faster than TITANIC. The computing time obtained by FP-TREE is 0.048 second to find the final result while for TITANIC is 0.066 second. The TITANIC algorithm begins to take over as the number of items increases. For sparse contexts, the TITANIC algorithm gives better results, and FP-TREE does not take up more memory space.

V. REFERENCES

[1]. Hamrouni, T. Extraction of generic informative bases of rules without calculation of closings. Retrieved from memoireonline: https://www.memoireonline.com/03/10/3241/m_ Extraction-des-bases-generiques- informatives-de-regles-sans-calcul-de-fermetures9.html

[2]. Jiawei Han et al, Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach, Data Mining and Knowledge Discovery, 8, 53–87, 200 4

[3]. Gerd Stumme et al, "Computing iceberg concept lattices with TITANIC", Knowledge Engineering 42 (2002) 189–222

[4]. K. Wang, Y. He, D. Cheung, Y. Chin, "Mining confident rules without support requirement", in: Proceedings of ACM International Conference on Information and Knowledge Management, CIKM, 2001, pp. 89-96.

[5]. H. Xiong, P. Tan, V. Kumar, "Mining strong affinity association patterns in data sets with skewed support distribution", in: Proceedings of the Third IEEE International Conference on Data Mining, ICDM, 2003, pp. 387-394.

[6]. Ya-Han Hu, Yen-Liang Chen, "Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism", Decision Support Systems, 2006, 42, pp. 1-24.

[7]. J. Ding, "Efficient association rule mining among infrequent items", Ph.D. Thesis, University of Illinois at Chicago, 2005.

[8]. Ling Zhou, Stephen Yau, "Efficient association rule mining among both frequent and infrequent items", Computers and Mathematics with Applications, 2007, 54, pp.

[9]. Y.Fakir et al, Extraction of itemsets frequents, International Journal of Mathematics Research. ISSN 0976-5840 Volume 12, Number 1 (2020), pp. 23-32

[10]. Youssef FAKIR et al, A Comparative Study between Relim and SaM Algorithms, International Journal of Computer Science and Information Security (IJCSIS),Vol. 18, No. 5, May 2020

[11]. Shamila Nasreen et al, Frequent pattern mining algorithms for finding associated frequent patterns for data streams: a survey, the 5th international conference on emerging ubiquitous systems and pervasive networks (EUSPN-2014)

[12]. Wael Mohamed et al, An implementation of Eclat on spark, International Journal of Computer Science and Information Security (IJCSIS),Vol. 15, No. 6, June 2017

Cite this article as :

Youssef FAKIR, Rachid El AYACHI, Mohamed FAKIR, "Mining Frequent Pattern by Titanic and FP-Tree algorithms", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 6 Issue 5, pp. 208-215, September-October 2020. Available at doi : https://doi.org/10.32628/CSEIT206537

Journal URL : http://ijsrcseit.com/CSEIT206537