Journal of Algorithms and Computation

journal homepage: http://jac.ut.ac.ir

BloomEclat: Efficient Eclat Algorithm based on Bloom filter

Sina Abbasi*[1] and Ali Moeini†[2]

[1,2]Department of Algorithms and Computation, School of Engineering Science, College of Engineering, University of Tehran, Iran.

ABSTRACT

Eclat is an algorithm that finds frequent itemsets. It uses a vertical database and calculates item's support by intersecting transactions. However, Eclat suffers from the exponential time complexity of calculating the intersection of transactions. In this paper, a randomized algorithm called BloomEclat based on Bloom filter is presented to improve the Eclat algorithm complexity in finding frequent itemsets. Through Bloom Filter, an elementâĂŹs membership to a set, can be checked and set operations such as intersection and union of two sets can be executed in a time efficient manner. By using these capabilities, Eclat algorithmâĂŹs intersecting problem can significantly improve. In BloomEclat algorithm with slight false positive error, the speed of the intersecting transactions is increased, and consequently the execution time is reduced.

Keyword: Eclat algorithm, bloom filter, frequent pattern mining, association rules mining, data mining, union, intersection.

ARTICLE INFO

Article history:
Research paper
Received 14, September 2020
Received in revised form 15, April 2021
Accepted 11 May 2021
Available online 01, June 2021

AMS subject Classification: 05C78.

*sina.abbasi.415@ut.ac.ir
†Corresponding author: A. Moeini. Email: moeini@ut.ac.ir

1 Introduction

The problem of frequent itemset discovery is finding the association rules in a market-basket model of data and also finding sets of items that appears in many of the same baskets. To distinguish the frequent itemsets from ordinary items the number of repetitions in the baskets must exceed a predefined minimum support S. If 'I' is a set of items, the size of support for âĂŸIâĂŹ equals the number of baskets that contain 'I'. So 'I' is frequent as long as it supports a value greater than âĂŸSâĂŹ. Finding the frequent itemset was initially proposed to analyze market-basket models, but over time it has become one of the methods of data mining in various domains, like bioinformatics [13], image classification [5], network traffic [6], and e-learning [12]. The algorithm for finding frequent itemsets was first introduced by R. Aggrawal [2]. Then, the Apriori [1] algorithm was introduced which suffers from a long execution time. Eclat algorithm was introduced to improve the performance of Apriori. However, if the number of transactions are high, then the operation of intersecting transactions become time-consuming. Normal intersecting uses a sequential comparison method of elements. In this paper, we use the Bloom Filter method, which calculates intersecting operation at a more reasonable time compared to the normal Eclat algorithm.
The rest of the paper is organized as follows: after discussing the Preliminaries in Section 2, Section 3, gives an illustration of the main algorithm, and in Section 4, we analyze the accuracy and time complexity of the proposed algorithm. Finally, Section 5, summarizes the results and proposes future research topics.

2 PRELIMINARIES

According to [4], algorithms seeking frequent itemsets can be divided into three separate categories:

2.1 Join-based algorithms

These algorithms use the bottom-up method for frequent itemsets mining and are used to generate larger itemsets as long as the itemsetsâĂŹ length is more than the minimum number of supports (defined by the user) in the database. The popular algorithm in this category is A-Priori. The algorithmâĂŹs name, Apriori, is because this method uses prior knowledge (prior step) to mine new frequent itemsets. It first creates a table with two columns. The first column contains items with length $k = 1$ and the second column holds the size of support for these items. If an itemâĂŹs support is less than the minimum support, it will be removed from the table. The same procedure continues for items with a length of $k + 1$ so that items with a length of $k + 1$ are formed by joining items with a length of k. And this occurs when the final joined itemâĂŹs support is less than the minimum (support). This algorithm often increases the speed of finding itemsets by removing items that are not frequent, but the number of $k + 1$ candidate items generated by joining the items of size k is very high. Through this algorithm for each k value, the database needs to be scanned once.

2.2 Tree-based algorithms

These algorithms use the set-enumeration concept to find frequent itemsets. By creating a lexicographic tree, the algorithm can search for itemsets in two ways: first level or first depth. The popular algorithm in this category is Eclat. Eclat [18] algorithm first converts the horizontal database to a vertical one with a single scan and mines frequent itemsets with enhanced performance compared to the Apriori method. Eclat directly calculates support by intersecting the transactions of each item with transactions of another one in the vertical database. If the two transactions $t(X)$ and $t(Y)$ are transactions of items X and Y, then we have:

$$t(XY) = t(X) \cap t(Y)$$

The size of support for XY itemsets equals $\mid t(XY) \mid$. The Eclat algorithm does not require sequential and costly database scanning and requires a single database scan. If the number of transaction is high, the cost of finding intersecting transactions will increase.

2.3 Pattern growth algorithms

These algorithms use divide and conquer algorithms for partitioning and projecting databases depending on frequent patterns that are already found, and Use these patterns for forming longer projecting databases. The popular algorithm in this category is FP-Growth algorithm, that [7] was introduced in 2000 by Han to improve Apriori algorithmâĂŹs performance by eliminating consecutive database scans. This method uses the divide and conquer algorithm. It initially compresses the database data into a tree called FP-Tree. The tree then becomes a conditional FP-Tree for each item so that the items can be explored separately. This will reduce searching expenses for frequent itemsets but will be time-consuming to build an FP-Tree database.
Eclat directly calculates support for candidate itemsets by intersecting transactions. This algorithm computes frequent itemsets of size $k+1$ by intersecting frequent itemsets of size k. The main step of this algorithm is to calculate the intersection of transactions. For a large transaction, this algorithm is inefficient. So, improving the transaction intersecting calculation is crucial.
Table 1 shows the Eclat algorithm.
In the algorithm above, The inputs are the minimum supports and for each $i \subseteq I$ is a set of frequent itemset $[i, t(i)]$ that have support values greater than or equal to the minimum support.
Our goal in this algorithm is to intersect the set of itemsets (tidsets) $X_a \in P$ with the set of other itemsets $X_b \in P$ so that $X_b \neq X_a$. Candidate itemset X_{ab} can be computed from the union of X_a and X_b (Line 6) to determine itâĂŹs frequency.
It is enough to intersect $t(X_a)$ with $t(X_b)$ (line 7). X_{ab} is then added to a new variable Pa that contains all itemsets with the prefix X_a. The Eclat function is called recursively

Algorithm 1. Eclat algorithm

Input: minimum support threshold minsup
Input $P \leftarrow \{(i, t(i)) | i \in I, |t(i)\ |, \geq minsup\}, F \leftarrow \emptyset$
Output : all frequent itemsets F

```
1  ECLAT($P$, minsup, $F$)
2  For each $(X_a, t(X_a))P$do
3     $F \leftarrow F \cup [(X_{\mathrm{a}}, \sup(X_{\mathrm{a}}))]$
4     $P_a \leftarrow \emptyset$
5     For each $(X_{\mathrm{b}}, \boldsymbol{t}(X_{\mathrm{b}})) \in P$, with $X_{\mathrm{b}} > X_{\mathrm{a}}$ do
6        $X_{ab} = X_a \cup X_b$
7        $t(X_{ab}) = t(X_a) \cap t(X_b)$
8        if $sup(X_{ab}) \geq$ minsup then
9        $P_a \leftarrow P_{\mathrm{a}} \cup [(X_{\mathrm{ab}}, t(X_{\mathrm{ab}}))]$
10         End if
11  End For each
12  End For each
13    if $P_a \neq \emptyset$ then ECLAT$(P_a, minsup, F)$
14    End if
```

and finds all items that have the prefix X_a. This process continues until X_a can no longer be expanded.
Eclat algorithm suffers from intersection exponential time complexity. For this reason, in recent years, various algorithms have been introduced to improve Eclat's performance, including the Diffset [18] algorithm, which only saves the difference between the item's transaction ID instead of keeping the transaction intersection. Other methods, such as bit operations [16], FPGAs [15], and Pruning [10], have been introduced to reduce the intersection calculation time of two transactions. Parallel algorithms have also been introduced based on MapReduce [20]. One of the recently introduced randomized algorithms to improve subscription speed is HashEclat [19], which uses the minhash technique to calculate transaction intersecting.

2.4 Bloom Filter

Bloom Filter [3] is a space-efficient probabilistic data structure for storing data. Using Bloom Filter, membership of an element to a set can be checked. Bloom Filter also allows two sets to intersect in a reasonable time. The first and base element of this randomized data structure is an m-sized bit array in which initially all elements are set to zero. The second element is a group of $'k'$ hash functions. The first operation in this data structure is add operation, where the k hash functions are calculated for input element x. Then, the bits at k indices of the base array corresponding to the output of the hash functions $h_1(x), h_2(x), \cdots, h_k(x)$ are set to one, So for each input value, the value of k bits in the base array of Bloom Filter changes from zero to one. The second operation is to find an element w in Bloom Filter. Separate hash functions are calculated for w and all corresponding indices are checked to be set to 1 in the base array. If all the bits are set, then it can be said that w is probably a member. If any of the bits at these indices is 0 then w is certainly not a member. It is said that âĂIJw is probably a memberâĂİ because this method has a false positive error. In this way, a Bloom member element may not be in filter, but it may be mistakenly identified as a member of a bloom filter. According to the following formulas, the optimal values of Bloom Filter (m) size, number of a hash function (k), and also false positive error value (FP) can be computed [11].

$$\mathrm{FP} = \left(1 - \left[1 - \frac{1}{\mathrm{m}}\right]^{\mathrm{kn}}\right)^{\mathrm{k}} \tag{1}$$

$$m = -\frac{\mathrm{n}\ln \mathrm{FP}}{(\ln 2)^2} \tag{2}$$

$$k = \frac{\mathrm{m}}{\mathrm{n}}\ln 2 \tag{3}$$

3 PROPOSED ALGORITHM

In this paper, we utilize Bloom Filter to calculate the intersecting of sets or lists at a more reasonable time compared to the normal Eclat algorithm.

3.1 Intersection with bloom filter

To find the intersection of two lists, two separate Bloom filters with a fixed size of m and k are constructed. Then a bitwise AND operation is done on these two bloom filters. A more efficient way is to enter one of the lists in a Bloom filter and check if each member of the second list is inside this Bloom filter or not. If the desired member is inside the Bloom filter, we save it in a new intersection list and if not, we check the membership of the other members. We do this for all members of the second list. The list *'Intersection'* is the result of intersecting these lists. In algorithm 1, line 7, the proposed method can be analyzed as follows:
First, we enter $t(X_a)$, which includes transactions containing X_a itemsets, into Bloom Filter B. Now we check each member of $t(X_b)$ for whether they are a member of Bloom Filter B or not. If any is a member, we save it in the '*Intersection*' list, otherwise, we proceed to the next member of $t(Xb)$.

3.2 Union with bloom filter

The Bloom Filter method can also be used for the union of two lists. One suggested method is to enter each of the lists in a separate Bloom Filter with a fixed size of m and

```
Intersection with Bloom Filter
1  Input: $Intersection \leftarrow \emptyset$
2  Output : Intersection of transactions
3  $B \leftarrow$ empty Bloom Filter of size $m$
4  For each $i \in t(X_a)$ do
5     $B.\textbf{Add}(j)$
6  End For each
7  For each $j \in t(X_b)$ do
8     if $j \in B$ do
9        $Intersection.\textbf{Add}(j)$
10    End if
11 End For each
```

k, and then bitwise union the two resulting Bloom Filters. A more efficient approach is to enter one of the lists in a Bloom Filter and check to see if each member from the second list is included in the Bloom Filter that carries the first list. If the desired member is inside the Bloom Filter, we proceed to another member and if not, we save it in a new '*Union*' list. We do this for all members of the second list. Then we save all the members of the first list in '*Union*'. The list '*Union*' is the result of intersecting these lists.
In the algorithm 1 in line 6, the proposed method can be analyzed as follows:

```
Union with Bloom Filter
1  Input: $Union \leftarrow \emptyset$
2  Output : Union of transactions
3  $B \leftarrow$ empty Bloom Filter of size $m$
4  For each $i \in X_a$ do
5     $B.\textbf{Add}(j)$
6     $Union.\textbf{Add}(j)$
7  End For each
8  For each $j \in X_b$ do
9     if $j \in B$ do
10       $Union.\textbf{Add}(j)$
11    End if
12 End For each
```

First, we insert the X_a itemsets into Bloom Filter B and add the X_a members to the '*Union*' list. Now we check for each member of X_b to see whether it is a member of Bloom Filter B or not. If it is, we go to the next X_b member, otherwise, we save it in the Union list.

4 EXPERIMENT RESULTS

The experiment was run in Python programming language on a Lenovo Ideapad 510 Intel Core i7-7500U CPU @ 2.70GHz 2.90GHz system with 12GB RAM in Win10 64-bit platform. To test the BloomEclat algorithm, we used the Mushroom dataset [8], which is one of the most popular datasets for working with frequent itemsets, The number of transactions in this database is 8123 and The dataset is of a dense type. the minimum support value is set to 10%.
The value of the Bloom filterâĂŹs input n should be set to equal the size of the data transactions, which is 8123. Because in a worst case scenario, an item will be present in all transactions and ItâĂŹs support size is equal to the size of all transactions. By doing this, we can find the most frequent itemsets of the densest datasets at a more efficient time compared to the normal Eclat algorithm, and and it wonâĂŹt effect the proposed algorithm that the dataset is dense or sparse.
Based on the optimal formulas for k, m, and FP, we need the value of m to find the optimal value of FP, and we also need to have the value of FP in hand to find the optimal value of m. thus we have two ways to examine the proposed algorithm [14]:
1) Let $n = 8123$, with the optimal default value of FP (which is equal to 0.001, which means 0.1%), we calculate the optimal value for m (which according to equation (2) is equal to 116790).
According to equation (1), (2), and (3), the optimal value for hash functions based on m and n is 10 ($k = 10$).

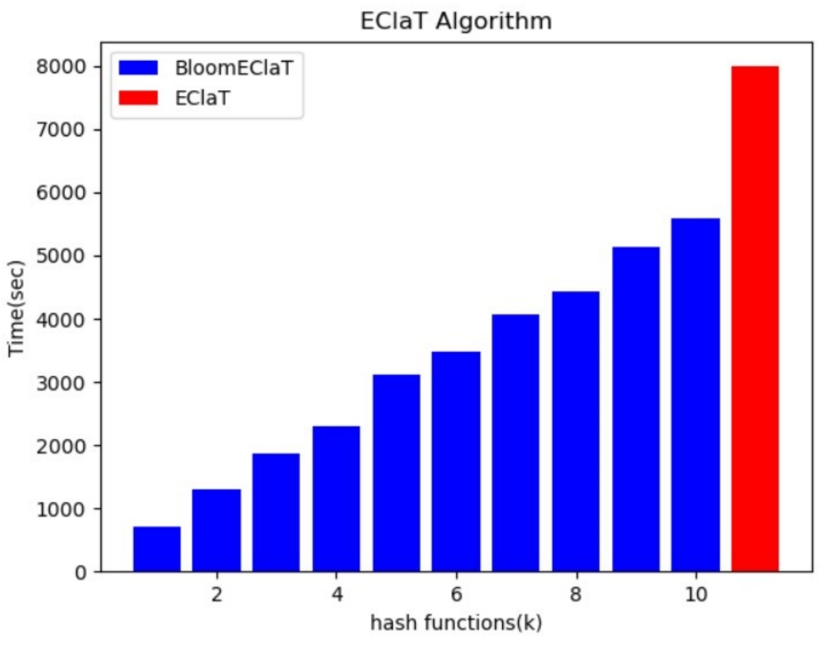

Figure 1: The comparison of the proposed method with the optimal size for m and for different values of k against the normal Eclat algorithm

Fig.1 shows the comparison of the proposed method with the optimal size of m and for different values of k against the normal Eclat algorithm. The red column in the fig 1 displays of normal eclat algorithm and the blue columns represent the proposed algorithm for different hash functions. the proposed algorithm in the worst case(for $k = 10$), runs faster than the normal eclat algorithm.
According to equation (3), the optimal value for k based on n and m is 10. For k less than 10, the FP value will increase according to equation (1).

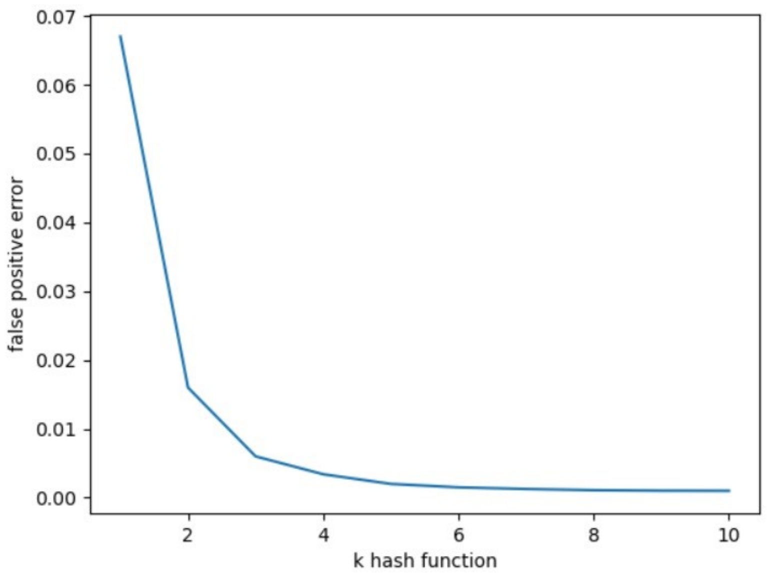

Figure 2: The value of FP for different k

Based on equation (1) the value of FP for different k is given in Fig.2.
Fig. 2 shows the FP value for different hash functions. based on equation (1), for optimal FP, if the number of hash functions is higher, the value of FP is shorter. But if the number of hash functions increases, the execution time of the algorithm will increases.
The accuracy of the proposed algorithm is also shown in Fig.3.
Fig.3 shows the accuracy for different hash functions. if the number of hash functions is higher, the accuracy gets higher. But if the number of hash functions increases, the execution time of the algorithm will increases.
2) In the next method, by assigning $n = 8123$ and fixing to $FP = 0.001$, we calculate the value of m for different values of k. The difference between this method and the above, is that the value of m is not optimal, but instead, for all values of k, the value of FP will be constant and equal to 0.001.
Fig.4 shows a comparison of the proposed method with the size $FP = 0.001$, for different values of k and m with the normal Eclat algorithm. Fig.1 shows the comparison of the proposed method with the optimal size of m and for different values of k against the normal Eclat algorithm. The red column in the fig.1 displays of normal eclat algorithm and the blue columns represent the proposed algorithm for different hash functions. the

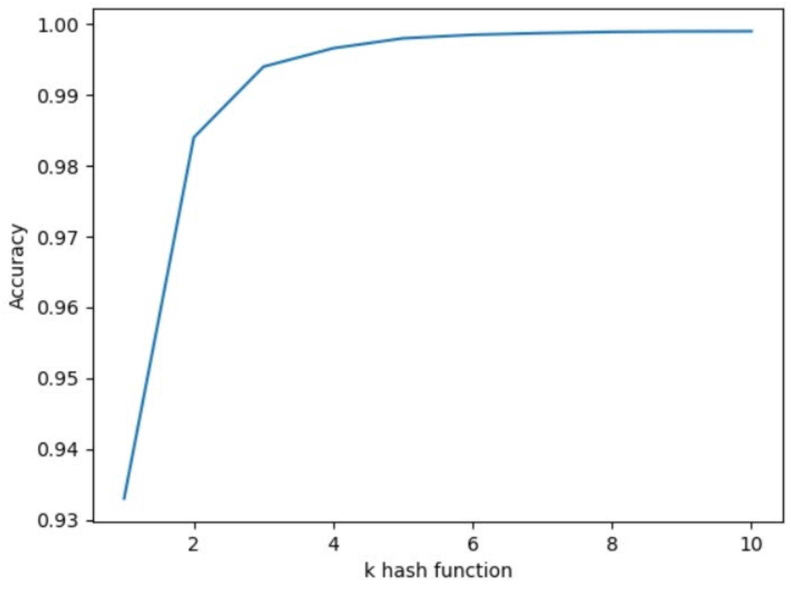

Figure 3: The accuracy of proposed algorithm for different k.

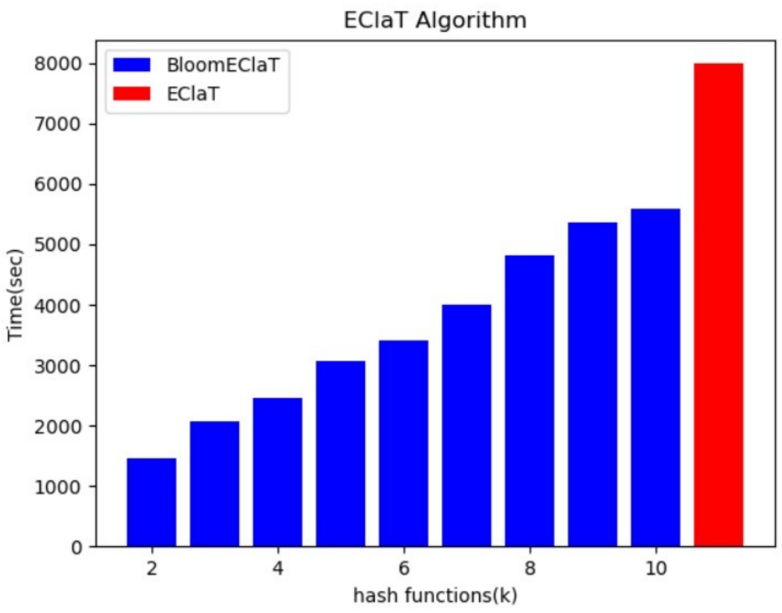

Figure 4: The comparison of the proposed method with the size $FP = 0.001$, for different values of k and m with the normal Eclat algorithm.

proposed algorithm in the worst case(for $k = 10$), runs faster than the normal eclat algorithm.
In method 1, the value of m, and in this method the value of FP is constant.

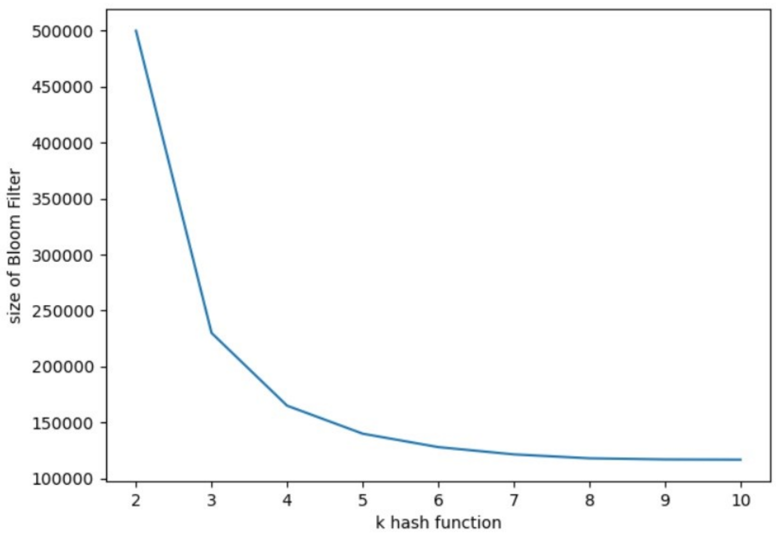

Figure 5: The values of m (size of Bloom Filter) for different values of k.

According to equation (2) the values of m for different values of k are shown in Fig.5. Fig.5 shows the FP value for different hash functions. based on equation (1), for optimal FP, if the number of hash functions is higher, the value of FP is shorter. But if the number of hash functions increases, the execution time of the algorithm will increases. The method 2 can be used when the amount of memory consumption is important.

5 CONCLUSION

In this paper, we proposed the BloomEclat algorithm, which uses the Bloom Filter data structure to improve the intersecting of the Eclat algorithm for mining frequent itemsets. The Eclat algorithm directly calculates support by intersecting the transactions of each itemset in the vertical database with the transactions of another itemset. To improve intersection and union in the Eclat algorithm, we entered the members of the first list into the Bloom Filter and checked if each second-list member is a member of the bloom filter or not. As our experimental results show, this method can intersect transactions with a small error percentage in less time compared to the normal Eclat algorithm. In future work, we will look at data that have very high number of items and to increase the union time in the Eclat algorithm. We will also look at the XNOR logic operator in the Bloom Filter method, which can be used for infrequent itemset mining [9].

References

[1] Agrawal, R. and Srikant R., (1994). Fast algorithms for mining association rules. Proc. 20th int. conf. very large data bases, VLDB.

[2] Agrawal, R., et al. (1993). Mining association rules between sets of items in large databases. Proceedings of the 1993 ACM SIGMOD international conference on Management of data.

[3] Bloom, B. H. (1970). "Space/time trade-offs in hash coding with allowable errors." Communications of the ACM 13(7): 422-426.

[4] Chee, C.-H., et al. (2019). "Algorithms for frequent itemset mining: a literature review." Artificial Intelligence Review 52(4): 2603-2621

[5] Fernando, B., et al. (2012). Effective use of frequent itemset mining for image classification. European conference on computer vision, Springer.

[6] Glatz, E., et al. (2014). "Visualizing big network traffic data using frequent pattern mining and hypergraphs." Computing 96(1): 27-38

[7] Han, J., et al. (2000). "Mining frequent patterns without candidate generation." ACM sigmod record 29(2): 1-12.

[8] http:// fimi.ua.ac.be/data

[9] Lu, Y., et al. (2018). Efficient infrequent itemset mining using depth-first and top-down lattice traversal. International Conference on Database Systems for Advanced Applications, Springer.

[10] Ma, Z., et al. (2016). "An improved eclat algorithm for mining association rules based on increased search strategy." International Journal of Database Theory and Application 9(5): 251-266.

[11] Mitzenmacher, M. and Upfal, E., (2017). Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis, Cambridge university press.

[12] Mwamikazi, E., et al. (2014). A dynamic questionnaire to further reduce questions in learning style assessment. IFIP International Conference on Artificial Intelligence Applications and Innovations, Springer.

[13] Naulaerts, S., et al. (2015). "A primer to frequent itemset mining for bioinformatics." Briefings in bioinformatics 16(2): 216-231.

[14] Pozo, M., et al. (2016). An item/user representation for recommender systems based on bloom filters. 2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS), IEEE.

[15] Shi, S., et al. (2013). Accelerating intersection computation in frequent itemset mining with FPGA. 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, IEEE.

[16] Xiong, Z., et al. (2010). "Improvement of ECLAT algorithm for association rules based on hash Boolean matrix." Application Research of Computers 4: 1323-1325.

[17] Zaki, M. J. (2000). "Scalable algorithms for association mining." IEEE transactions on knowledge and data engineering 12(3): 372-390.

[18] Zaki, M. J. and Gouda K., (2003). Fast vertical mining using diffsets. Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining.

[19] Zhang, C., Tian, P., Zhang, X., Liao, Q., Jiang, Z.L. and Wang, X., 2019. HashEclat: an efficient frequent itemset algorithm. International Journal of Machine Learning and Cybernetics, 10(11), pp.3003-3016.

[20] Zhang, Z., et al. (2013). MREclat: an algorithm for parallel mining frequent itemsets. 2013 International Conference on Advanced Cloud and Big Data, IEEE.