Received: December 3, 2022. Revised: February 21, 2023.

INASS

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

A Logic Design-based Approach for Frequent Itemsets Mining Using LCO Algorithm

Oday Ahmed Al-Ghanimi[1]* **Hussein K. Khafaji[2]**

[1] *Informatics Institute for Postgraduate Studies, Baghdad, Iraq*
[2] *Al-Rafidain University College, Baghdad, Iraq*
* Corresponding author's Email: ms202030612@iips.icci.edu.iq

Abstract: Frequent itemsets (FIs) mining is the main challenge in analyzing association rules since the complexity of association rule mining is determined mainly by identifying all frequent itemsets. There are many approaches to FIs mining; each approach includes many algorithms, but there is a major weakness in the logic-design-based FIs approach because there is a limited number of methods for logic circuit or expression optimization, each of which suffers from some drawbacks that are transferred to the mining process when they are utilized for this purpose. We propose LCOFI, a new algorithm for frequent itemset mining based on the LCO algorithm, in this paper. LCOA is a new algorithm for logic circuit optimization. LCOFI utilizes the suggested operations for a bipartite graph in LCOA. The proposed algorithm is simple and efficient and supports a large number of input items. It scans the transaction database only once to construct the bipartite graphs of frequent 1-itemsets and avoids the generation of candidate itemsets. It does not require complex data structures such as trees and hash tables to validate the frequency of FIs. When applied to a variety of datasets with varying characteristics, the proposed algorithm, LCOFI, outperformed the Apriori and Fp-growth algorithms in all the experiments.

Keywords: Data mining, Association rule, Frequent itemset, Bipartite graph, LCOA.

1. Introduction

One of the well-known steps in the process of knowledge discovery in the database, KDD, is data mining [1]. Mining association rules is one of the most important tasks of data mining [2, 3]. Due to the importance of association rules and their mining difficulties, many algorithms have been developed for AR mining. The Apriori algorithm is frequently utilized in discovering association rules, and most algorithms that are utilized to scan candidate itemsets depend on it [4]. The AR mining process can be divided into two steps; 1) mining all frequent itemsets and; 2) extracting trustworthy association rules from the mined frequent itemsets [5]. The essential step of association rules mining is frequent itemsets mining; it is NP-complete problem [6-8]. The problem of mining frequent itemsets was first proposed by R. Agrawal, T. Imielinski, and A. N. Swami [9]. There are several FIs mining algorithms. These algorithms can be divided into four approaches:

- Apriori-based approach,
- lattice-based approach,
- graph-based approach, and
- logic circuit design-based approach.

In the first approach, R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo [10] produced the Apriori algorithm, and then many algorithms are proposed for FIs mining derived from it. These algorithms are not entirely different from the state of the art algorithm, Apriori. The Apriori algorithm inspired many researchers to develop it or to design an algorithm that comes close to its properties.

In the second approach, i.e., the lattice-based approach, D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, and T. Yiu [11] produced MAFIA, a novel

algorithm for mining maximal frequent itemsets from a transactional database. When the maximal itemsets have been mined, all frequent itemsets can be derived from them. In the same approach, M. J. Zaki. [12] developed CHARM, an efficient algorithm for mining closed frequent itemsets, and [13] CHARM-L, an efficient algorithm for creating the closed itemset lattice (CIL).

One of the main algorithms that depend on the graph principles is the FP-Growth algorithm. V. Tiwari, V. Tiwari, S. Gupta, and R. Tiwari [14] developed what so-called the FP-growth-graph algorithm, which organizes items for mining frequent itemsets using a graph rather than a tree or lattice. The advantage of using a graph is that each item only has one node which leads to less memory consumption. Kumar, Srinivasu, and Ch, N.D. [15] proposed graph-based frequent patterns mining approach that generates patterns without generating candidate itemsets unlike traditional approaches such as Apriori. The advantage of a graph-based algorithm is that it consumes less time of generating frequent patterns than traditional methods.

Many researchers utilized the similarity between the binary representation of transactional databases and truth tables. In both these tables, 1 means the existence of an item in a transaction of a transactional database or the truth value of a variable in a logical sum of product, SOP. This fact leads to the fourth approach, i.e., the logic circuit design/minimization-based approach, which depends on the methods of logical circuit/expressions minimization to mine frequent itemsets or minimize the mined association rules. The algorithms in this approach do not achieve distinguished success because they inherit the drawbacks of the methods of the logic design or because they suffer from weak points resulting from the incomplete matching between the FI mining, FIM, problem and the logic circuit design. So let's consider the three most famous algorithms for logic circuit design, which are the Karnaugh map or K-map [16], Quine-McCluskey Q-M [17], and Espresso [18], with the three algorithms that emerged from them to mine association rules, with a brief description of the drawbacks inhered from the former algorithms:

- In order to enhance the digital logic circuit based on the truth table, the Karnaugh map, or K-map, was developed. K-map has flaws that have been discovered through numerous studies; it is difficult to use and implement as software [19]. Also, it is very unclear when a problem contains more than four variables [20]. Sharma and Singh. [21] proposed a K-Partition algorithm for frequent patterns mining which is an advancement in the traditional partition algorithm. This algorithm depended on the K map method of logic circuit design. The K-map-based association rule mining algorithms have the restrictions that they could only function effectively with a maximum of four laterals/items. A drawback of the K-map-based algorithm is that it is not effective in discovering all frequent itemsets with more than four literals/items.
- The tabular or Quine-McCluskey Q-M approach, the first substitute for K-Map, was created by Edward McCluskey and Willard Quine. The set of minimum prime implicants emitted by the output functions [22] is determined by a methodical process that starts with a truth table and concludes with it. Although the Quine McCluskey approach is amenable to automation as a computer program, it is inefficient in terms of execution time and memory usage, so that adding one additional literal will practically quadruple these two consequences of the reduction cost [23]. In conclusion, the Quine McCluskey method is far from understandable and visually appealing [24, 17], yet it is effective for a small number of input literals and output functions. Khedr, Ramadan, and Abdel-Magid [25] proposed a new algorithm Quine-McCluskey Rule (QMR) for association rule minimization in the dataset depending on Quine-McCluskey (Q-M) algorithm Logic circuits Optimization Technique. The drawbacks of this algorithm are that it deals with a limited number of inputs, and therefore the process of reducing the association rules is very expensive in terms of time and space.
- Brayton et al. [26] created the ESPRESSO algorithm, which uses very few computer resources while providing excellent performance. Because ESPRESSO iterates to manipulate "cubes" representing product terms, its minimized output is not guaranteed to produce optimal minimization, in addition to its reliance on vector optimization, making it a loser in terms of visualization and ease of understanding [18]. Ashmouni, Ramadan, and Rashed. [27] proposed an approach for reducing association rules in the dataset depending on the espresso algorithm for rule minimization. A drawback of the espresso-based algorithm is that the rules resulting from this algorithm are not in the optimal minimization.

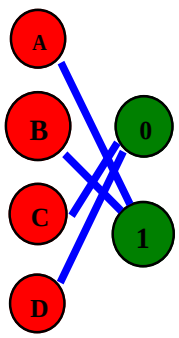
The aim of this paper is to avoid the drawbacks of the algorithms that can be listed within the logic design-based approach for FIM by selecting suitable algorithm for logic design to be adapted for the FIM problem.

O. A. Al-Ghanim and H. K. Khafaji, 2022 [28] proposed LCOA, a novel algorithm that relies on bipartite graph properties and suggests graph operations to optimize logic circuits and expressions. This algorithm is easy to implement as a program; it can deal with an unlimited number of literals and variables; and it is simple to comprehend and visualize.

To avoid the drawbacks of the algorithms that can be listed within the logic design-based approach for FIM, this paper presents a new algorithm to mine FIs depending on LCOA.

The proposed algorithm will contribute to the FIM problem in the following directions:

1. utilization of the similarity between FIM and logic design problems, which leads to empowering the logic circuit design-based approach by designing a new algorithm for FIM problem depending on LCOA. Solving similar problems with similar algorithms definitely reduces the design steps and complexity and provides enough duration to select a suitable algorithm of a given problem to be modified to manipulate another.
2. The proposed algorithm scans the database only once by selecting the frequent 1-itemsets, and all the required operations, such as support counting and frequent k-itemsets generation, are accomplished on-the-fly, which helps maintain the memory consumption.
3. The proposed algorithm does not need a special data structure such as hash trees or an FP-tree to generate and prune candidate itemsets, but it uses a bipartite graph to represent the itemsets.
4. It lends itself to parallelism because each item in a database can be manipulated independently. This feature is not considered in this paper.
5. The above characteristics ensure that the FIM's speed can be accelerated by using a single computation machine or a parallel computing machine.



	0	1
A	0	1
B	0	1
C	1	0
D	1	0

Figure. 1 Representation of the PBG ABC'D'

The proposed algorithm will be explained in detail in section 3. The organization of the paper is given as follows: the preliminaries of this research are given in section 2. The explanation of the proposed algorithm is illustrated in section 3. The experiment datasets are given in section 4. The results and discussion are illustrated in section 5 and the conclusion of this research work is given in section 6.

2. Preliminaries

In this section, we discuss basic concepts related to the proposed algorithm. These concepts include the definition of frequent itemsets, the logic circuit optimization algorithm, LCOA, the bipartite graph, and the product bipartite graph, PBG. The purpose of presenting these concepts is their role in designing the proposed algorithm to mine frequent itemsets that are hidden in transaction databases.

2.1 Frequent itemset mining

Let I = $\{I_1,..., I_n\}$ be a collection of literals, called items. Let D be a collection of transactions, where each transaction T is a set of items such that T $\subseteq$I and each transaction are associated with a unique identifier called TID. All items included in the transaction are listed in chronological order. An itemset A is a set of items in I and is known as a k-itemset if it contains k items from I. An itemset A is regarded as a frequent itemset if its occurrences (support) in D are greater or equal to a min_sup threshold that is determined by the user, and it's regarded as an infrequent itemset otherwise [29, 30].

2.2 Bipartite graph

In the graph theory field, a bipartite graph (or bigraph) is a graph whose nodes' set, N, can be partitioned into two disjoint and independent sets; N_0 and N_1. Every edge e in the edge set E links a node in N_0 to one node in N_1 [31]. The node sets $N_0 = \{n_{01}, ...n_{0n}\}$ and $N_1=\{n_{11}... n_{1m}\}$ are the mutually exclusive vertices sets and they are called the graph's parts [31, 32]. E $\subset N_0 \times N_1$ is a set of edges that connect vertices between two partitions[32, 33]. For example, Fig. 1 presents a bipartite graph with two sets of vertices N_0={A, B, C, D} and N_1={0, 1} where E={(A, 1), (B, 1), (C, 0), (D, 0)}, in addition to its biadjacency matrix [31] which includes four ones indicating the elements of E.

2.3 Product bipartite graph (PBG)

O. A. Al-Ghanim and H. K. Khafaji [28]

produced what is called "PBG." PBG is a term in an SOP for logical expression constructed as a bipartite graph. Therefore, the PBG graph consisted of two sets of nodes {0, 1} and the set of literals in the expression. For example, Figure 1 depicts the representation of the term ABC'D', i.e., the 13th entry of a truth table, such that the output of a logic circuit or expression is 1 when the value of A and B is 1 and the value of C and D is 0.

2.4 LCO algorithm

The logic circuits optimization algorithm consists of the following general steps [28]:

Step#1: Depending on the truth table or the number of true combinations construct the SOP expression.
Step#2: Construct the biadjacency matrices of each PBG in the SOP expression.
Step#3: Select two PBGs of the SOP expression PBG_i and PBG_j such that the number of edges of PBG_i is less than or equal to the number of edges of PBG_j, ($|E(PBG_i)| \leq |E(PBG_j)|$), (the inclusion property), and they have only one different edge. The process of elimination occurs in the larger PBG with keeping the resulting PBG and the smaller PBG, or the elimination process occurs on both PBGs in the case of equality and keeping the resulting PBG and neglecting the main PBGs.
Step#4: Repeat step 3 until the SOP is optimized.

In [28] the researchers suggested the XORing, ANDing, and ORing operations that can be done on the bipartite graphs of the PBGs. The proposed algorithm for mining frequent itemsets in this paper utilizes the inclusion property of two PBGs and these suggested logical operations are dedicated to PBGs in LCOA.

3. The proposed algorithm

In this section, we describe the proposed algorithm with examples. The proposed algorithm is an LCOA-based frequent itemset mining algorithm; for simplicity, it is abbreviated as LCOFI. It deals with three types of bipartite graphs:

- transaction database bipartite graphs,
- itemset-TIDset bipartite graphs, and
- itemset representation bipartite graphs.

Table 1. Transaction database

TransactionTID	Itemsets
TID1	a,b,c,d,e
TID 2	b,c,d,e
TID 3	b,c,e
TID 4	b,c,d
TID 5	a,b
TID 6	a,b,c, e

Table 2. Binary itemized representation of transaction database

Tid / Item	1	2	3	4	5	6
a	**1**	**0**	**0**	**0**	**1**	**1**
b	**1**	**1**	**1**	**1**	**1**	**1**
c	**1**	**1**	**1**	**1**	**0**	**1**
d	**1**	**1**	**0**	**1**	**0**	**0**
e	**1**	**1**	**1**	**0**	**0**	**1**

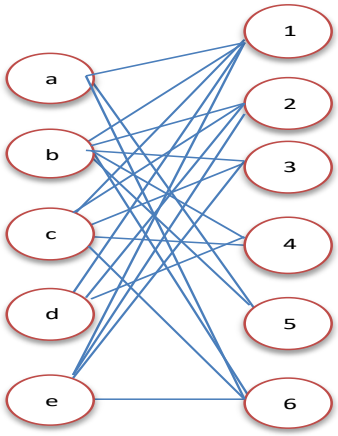
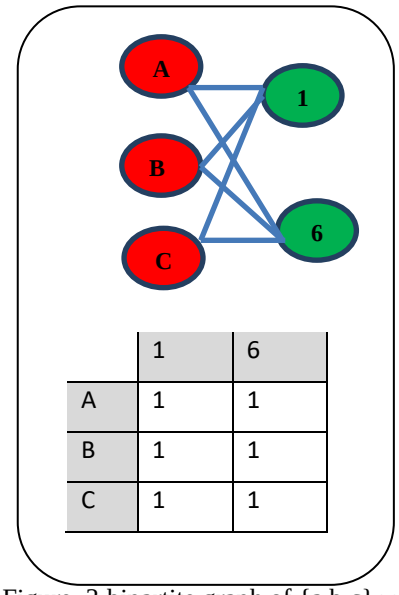


Figure. 2 Transaction database bipartite graph



	1	6
A	1	1
B	1	1
C	1	1

Figure. 3 bipartite graph of {a,b,c}↔{1,6}

Table 3. FIs mining processes and their counterpart in LCO algorithm

Seq.	Process	Suggested LCOA Operation
1	Selecting two k-itemsets i and j to generate a (k+1)-itemset such that i and j have only one different item.	*Using of equality inclusion property* of PBGs depending on XORing of two PBGs
2	Joining of the selected two k-itemsets i and j to generate a (k+1)-itemset	ORing of the itemset representation bipartite graph of itemset i and j using PBG ORing
3	Complex procedures to verify the frequentness of the generated (k+1)-itemset.	ANDing of itemset-TIDset bipartite graphs of i and j itemsets and compute the fun-out of the (k+1)-itemset.

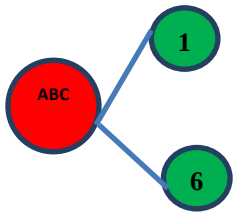


Figure. 4 Abstracted bipartitegraph of {a,b,c}↔{1,6}

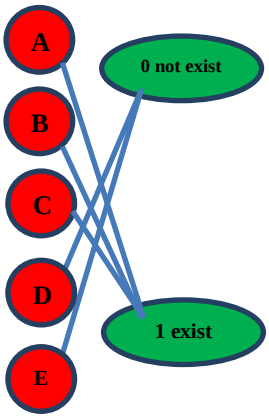


Figure. 5 SOP representation of ABC itemset using bipartite graph

For more explanation, consider Table 1, which includes an illustrative database consisting of six transactions and five items. Table 2 represents an itemized binary representation of the database. The first row indicates that item a is available in TIDs 1, 5, and 6. Fig. 2 presents a bipartite graph representation of the database. The notation *{itemset}↔{TIDset}* will be used in this paper, which means that the "itemset" is included in the transactions listed in the "TIDset." Consider the itemset {a, b, c}↔{1, 6}, which can be represented as a second type of bipartite graph (Fig. 3), or abstractly as Fig. 4. Indeed, the itemset ABC is similar to a term in an SOP of logic expression and can be written as ABCD'E', which means that the items a, b, and c exist in a transaction or an itemset but d and e are not available. This leads to the third type of bipartite graph in this context, as shown in Fig. 5.

Generally, the FIs mining algorithms require the processes listed in Table 3. The implementation of these processes varies from one algorithm to another. So, Table 3 includes the suggested operation for each process depending on LCOA to be adopted in the proposed algorithm LCOFI.

For more explanation, reconsider the database of Table 1. Suppose that the F_3, the set of frequent 3-itemsets, includes BCD and BCE according to min_sup value equals 2. Their PBGs graph is presented in Fig. 6. They are selected for the next generation to generate BCDE because they include only one different item. Determining the validity of these itemsets in LCOFI will depend on the XORing of their PBGs. The resultant PBG includes two items connected to the existing node which indicates the validity of the two itemsets, BCD and BCE, to play a role in the next generation. To accomplish the joint process to generate the itemset BCDE, a PBG ORing operation is required as shown in Fig. 6. To specify the frequentness of the Itemset BCDE, the itemset-TIDset bipartite graphs of BCD and BCE are ANDed as shown in Fig. 7.

The number of remaining transactions represents the support of the BCDE, i.e., 2 transactions 1 and 2. From the implementation point of view, the BPG is represented in abstracted PBG, recall Figs. 4 and 6, because all items have the same connections. Therefore the support counting of an itemset in LCOFI is just its fun-out in the graph.

After this preliminary information, it is possible now to present LCOFI. Algorithm 1 represents the general structure of the LCOFI algorithm. To provide more explanation, Table 4 presents a description of the functions and variables used in the proposed algorithms in addition to the explanation of the most steps in this paper.

The input for the algorithm is a transactional database, D. D is stored as binary values: 1 for an item presented in a transaction and 0 if the item is not presented. The second input is the minimum support, min_sup, while the output is the set of all

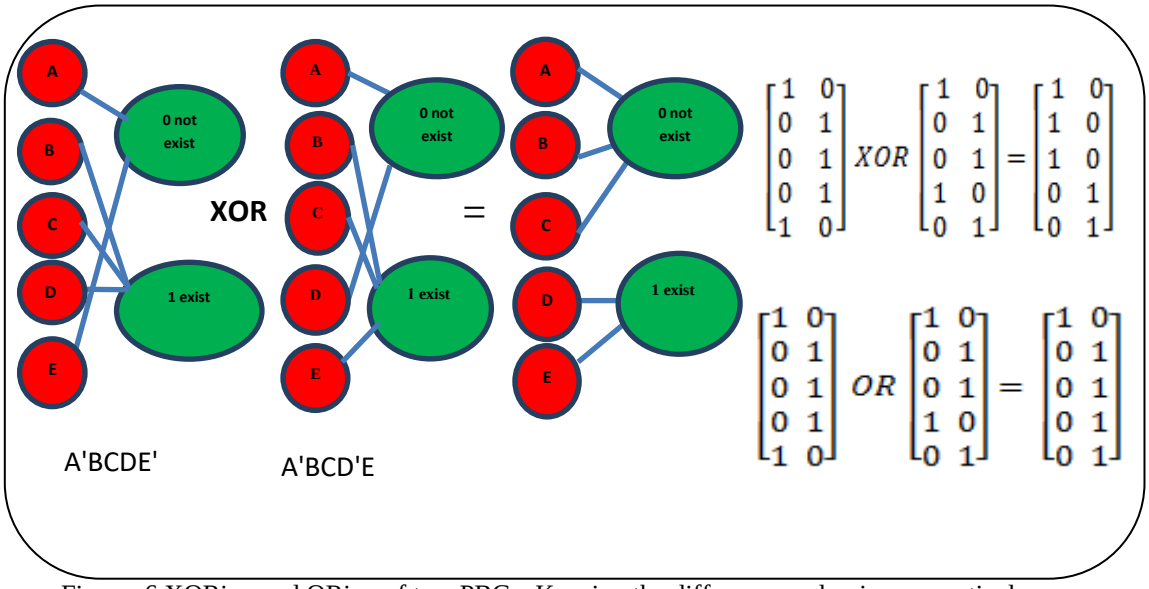
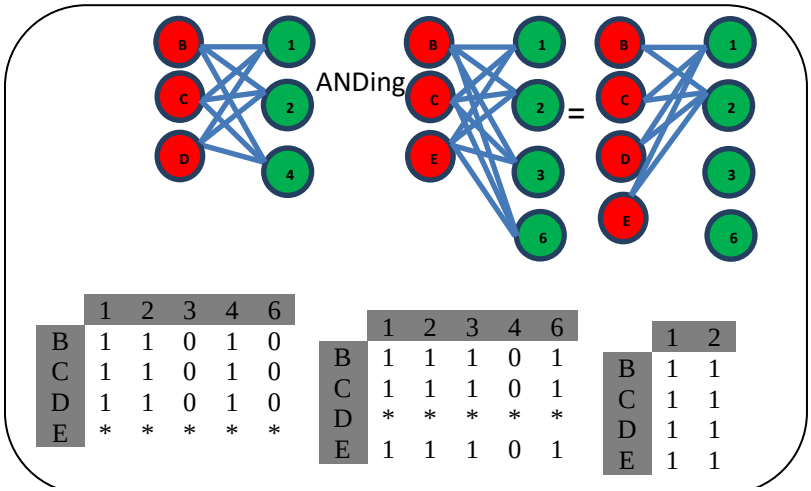


Figure. 6 XORing and ORing of two PBGs: Keeping the difference and union respectively



	1	2	3	4	6
B	1	1	0	1	0
C	1	1	0	1	0
D	1	1	0	1	0
E	*	*	*	*	*

	1	2	3	4	6
B	1	1	1	0	1
C	1	1	1	0	1
D	*	*	*	*	*
E	1	1	1	0	1

	1	2
B	1	1
C	1	1
D	1	1
E	1	1

Figure 7. ANDing operation of two PBGs to validate the frequentness of the generated PBG

frequent itemsets. Steps #1 and #2 are initialization steps. Step #3 is a call for an algorithm to generate 1-itemsets, it is called construct-1-itemsets(). The construct-1-itemsets() algorithm is presented in Algorithm 2. Step #4 represents a call for an algorithm to generate 2-itemsets; it is called construct-2-itemsets (F1). Construct-2-itemsets() are presented in Algorithm 3. Step #5 is the iteration process to call the construct-k-itemset(F_{K-1}) presented in Algorithm 4. This process is repeated to generate a set of all frequent itemsets, as shown in Steps #8 and #9. Step #11 will add the previously mined frequent itemsets to the set of frequent itemsets and return the result to the caller.

Algorithm 1. The proposed algorithm, LCOFI
Input:transaction dataset D; // Binary Representation
Min_sup; // (minimum support)
Output: Frequent-itemsets // set of frequent itemsets.
{
1. F_1={}; frequent-itemsets={};
2. K=4;
3. F_1=construct-1-itemsets(D);
4. F_2= construct-2-itemsets(F_1);
   frequent-itemsets=frequent-itemsets $\cup$ F_1 $\cup$ F_2;
5. while $F_{k-1}\neq${}
6. {
7. F_k=construct-k-itemset(F_{k-1});
8. frequent-itemsets=frequent-itemsets $\cup$ F_k;
9. k=k+1;
10. }
11. return frequent-itemsets;
}

Table 4. Description of the functions and variables used in the proposed algorithms

Seq.	Name of function or variable	Type	description
1	D	database	Binary transaction database
2	Min-sup	variable	Minimum support threshold
3	Frequent-itemsets	variable	Holds all the mined FIs
4	F_1	variable	Stores frequent items, i.e., frequent 1-itemsets
5	F_2	variable	Stores frequent items, i.e., frequent 2-itemsets
6	F_k	variable	Stores frequent items, i.e., frequent k-itemsets, (K>2)
7	f	variable	Represents an itemset in F_n where n>0
9	Construct- 1-itemsets	Function	To generates 1-itemsets
10	Construct-2-itemsets	Function	To Generates 2-itemsets
11	Construct-k-itemsets	Function	Generates k-itemsets
12	K	variable	Loop control variable and it represents the size of an itemset.
13	f-PBG	variable	Holds PBG of f as a biadjacency matrix
14	$f\text{-}PBG_{item}$	variable	Holds the item of the PBG of f.
15	$f\text{-}PBG_i$	variable	Holds PBG of i^{th} f as a biadjacency matrix
16	Anding, oring, and xoring	functions	Functions to do two matrices' AND, OR, or XOR operation respectively.
17	Fun-out	function	To count number of links of emerged from a node used to calculate the support of an itemset.
18	inclusion	function	To determine the equality-inclusion of two PBGs, i.e., itemsets.

3.1 1-itemsets PBGs construct

The pseudo-code of the algorithm to generate 1-itemsets is presented in Algorithm 2. The input for Algorithm 2 is a transactional dataset D and the min_sup threshold, while the output is a set of frequent 1-itemsets, F_1. It scans the transaction database D only once by constructing a bipartite graph, PBG_i, and then computes the support count of each item i by calculating its fun-out of PBG_i. This process is repeated to construct F_1, the list of frequent 1-itemsets, according to the min_sup threshold.

Algorithm 2. Construct-1-itemsets

```
   Input  : transaction dataset D;
            min-sup;
   Output : F1
1. {
2.          F1={};
3.          For each item  i  in D
4.          {
5.            construct PBGi;
6.            Support(i)=fun-out(i) ;
7.            If ( Support(i) >= min-sup)
8.                    F1= F1∪{i};
9.          }
10.           return F1;
11.  }
```

Step #2 is the initialization step. Step #5 constructs the PBG_i for each item i in D as a bipartite graph. The bipartite graph in this context includes a set of transactions in database D, including i, and a set of items, which includes one item i in this step. Simply, the PBG in this context will be expressed as {set of items}↔{set of transactions}, for example {a}↔{1, 5, 6} means that the item "a" is available in transactions 1, 5, and 6. For more explanation, consider Fig. 8.

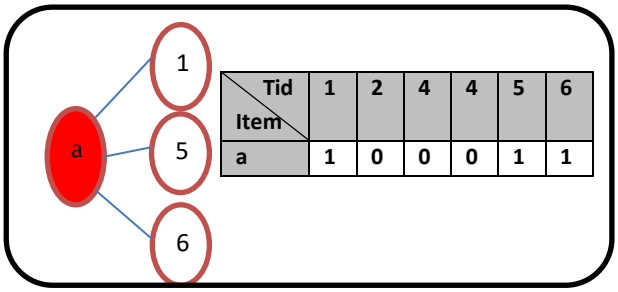


Figure 8. 1-itemset/PBG representation

The two sets, i.e., set of items and set of transactions, in PBG are joined by edges which represent the relation "the item i is available in the specified transaction set". These edges represent the occurrences of the items in the transaction database, therefore, the number of edges represents the fun-out which equals the support of item i according to the entered min_sup, this process is accomplished in step#6. The number of fun-out in the transaction set is the support of each item i in D, which is computed and checked according to the min_sup threshold in step#6. In steps #7 and 8, the item i will be added to the frequent 1-itemsets, F_1list, which is returned to the caller when the iteration terminates.

Example 1. Re-consider the database shown in Tables 1, 2, and Fig. 2. It can be equivalently represented as a bipartite graph, as shown in Table 2. which can be described as:

G=(V,E) where V={V1={a, b, c, d, e}, V2={1,2,3,4,5,6}} and E={(a, 1), (a, 5), (a, 6), (b, 1), (b, 2), (b, 3), (b, 4), (b, 5), (b, 6), (c, 1), (c, 2), (c, 3), (c, 4), (c, 6), (d, 1), (d, 2), (d, 4), (e, 1), (e, 2), (e, 3), (e, 6)}. For simplicity and abstraction, the grouping of items will be used to represent the bipartite graph as follows E={({a}↔{1, 5, 6}), ({b}↔{1, 2, 3, 4, 5, 6}, ({c}↔{1, 2, 3, 4, 6}), ({d}↔{1, 2, 4}), ({e}↔{1, 2, 3, 6})}. Suppose that the minimum support is 33%, i.e., ≈ 2 out of 6 transactions of the datasets presented in Tables 1 and 2. Consider the 1-frequent itemset presented in Table 2. The fun-out of {a} is three due to the availability of the links directed to the set of transactions {1, 5, 6}and this value is greater than 2, hence {a} is regarded as a frequent 1-itemset. In the same manner, the construction of PBGs and support counting is accomplished for the rest of the items which results in F_1={a, b, c, d, e}. These items have 3, 6, 5, 3, and 4 support counts respectively.

3.2 2-itemsets BPGs construct

After generating the frequent 1-itemsets presented in section 3.1, the next step is generating 2-itemsets depending on the contents of F_1. Algorithm 3 shows the process of generating frequent 2-itemsets from frequent 1-itemsets depending on the XORing, ORing, and ANDing operations proposed for PBGs.

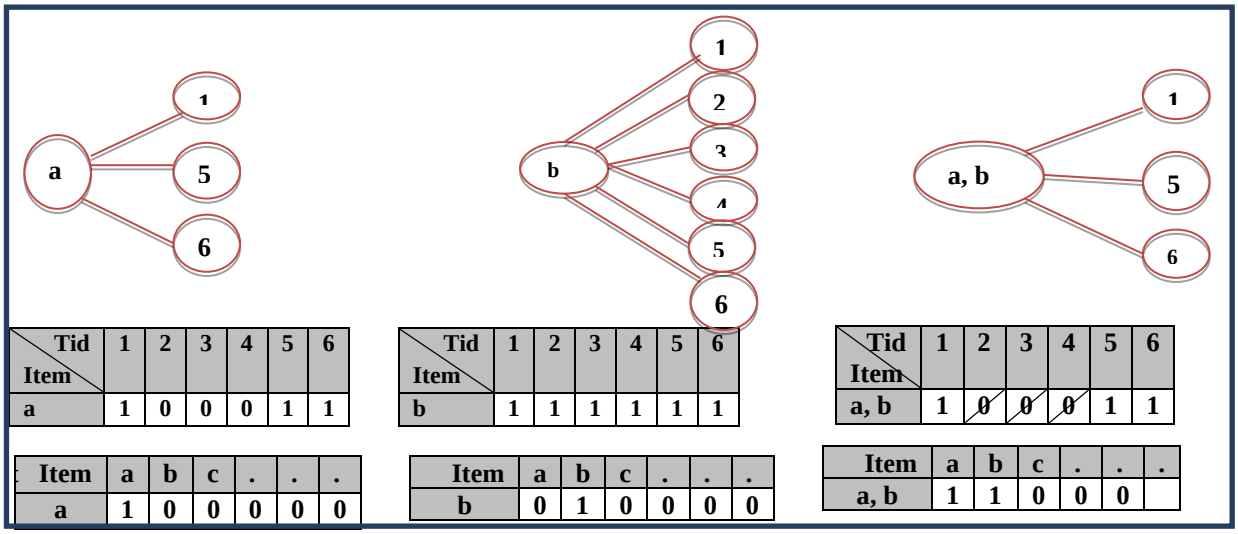
Step #4 picks the first frequent 1-item from the input F_1 list. Step#7 makes the ORing operation of the two frequent 1-itemsets, two items, to generate a 2-itemset, i.e., it takes the first 1-item, i.e., (f-PBG represented as a bipartite graph) from the F_1 list with each of the rest of the list items, i.e., (f_i-PBG) using the OR operation. Step#8 performs the ANDing operation of two tidlists of the combined frequent 1-itemset (f-PBG) from the ORing operation to find the tidlist of the 2-items, then the fun-out is computed to specify the min_sup of the generated 2-itemset. The number of fun-out in the (f-PBG) according to the itemset representation bipartite graph is the support of (f-PBG), which is computed and checked according to the min_sup threshold, this process is accomplished in step#9. In Step#10 the frequent 2-itemset will be added to the F_2 list. The above steps are repeated to construct the rest of the frequent 2-itemsets F_2. For more explanation consider Fig. 9.

Algorithm 3. Construct-2-itemsets(F1)

Input: F_1
Output: F_2
1. {
2. While $F_1 \neq \{\}$
3. {
4. remove f from F_1
5. for(i=1; length (F_1)>0; i++) do
6. {
7. f-PBG_{item} =oring(f-PBG_{item}, f_i- PBG_{item});
8. f-PBG=anding(f-PBG, f_i-PBG);
9. if (fun-out (f-PBG)>=min-sup)
10. F_2= $F_2 \cup$f-PBG
11. }// for
12. }// while
13. return F_2
14. }

The ORing of the PBG template and b PBG template, (100000) OR (010000), implies {a, b} PBG template, i.e., (110000). The ANDing of a and b PBGs, (100011) and (111111), result in (100011), which means ({a, b}↔{1, 5, 6}).

Consider example 1 and the following frequent 1- itemsets, F1 list generated from the previous step in section 3.1 which are F_1={a, b, c, d, e}. Fig. 9 includes steps of constructing PBGs for two sets of frequent 1- itemsets as sub bipartite graphs from the F_1 list presented in Table 2. To generate frequent 2-



Item \ Tid	1	2	3	4	5	6
a	1	0	0	0	1	1

Item	a	b	c	.	.	.
a	1	0	0	0	0	0

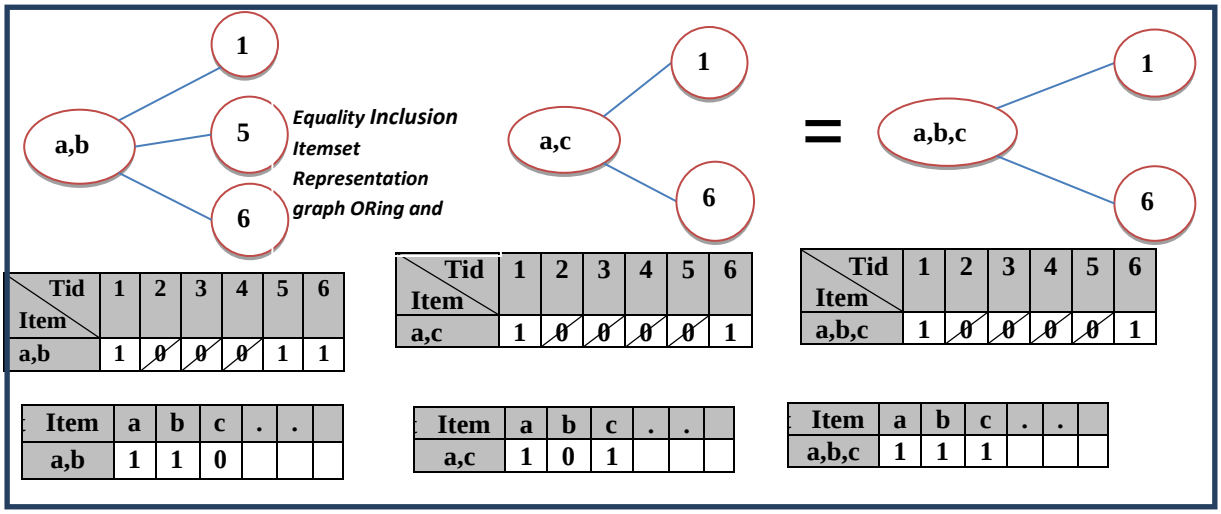
Item \ Tid	1	2	3	4	5	6
b	1	1	1	1	1	1

Item	a	b	c	.	.	.
b	0	1	0	0	0	0

Item \ Tid	1	2	3	4	5	6
a, b	1	0	0	0	1	1

Item	a	b	c	.	.	.
a, b	1	1	0	0	0	

Figure. 9 ORing of a and b item-template and their PBGs ANDing



Item \ Tid	1	2	3	4	5	6
a,b	1	0	0	0	1	1

Item	a	b	c	.	.	
a,b	1	1	0			

Item \ Tid	1	2	3	4	5	6
a,c	1	0	0	0	0	1

Item	a	b	c	.	.	
a,c	1	0	1			

Item \ Tid	1	2	3	4	5	6
a,b,c	1	0	0	0	0	1

Item	a	b	c	.	.	
a,b,c	1	1	1			

Figure. 10 ORing {a,b} and {a,c} item-template and their PBGs ANDing

itemset depending on the OR and AND operations proposed for PBGs. Let's trace the process of generating a frequent 2- itemset shown in Fig. 9.

Pick {a} as the first item from F_1 with each item of {b, c, d, e}. This process can be described simply as the combination of items by making OR operation template, i.e., ({a, b},{a, c},{a, d},{a, e},{b, c},{b, d},{b, e},{c, d},{c, e},{d, e}) and making AND operation of the transaction TID lists, i.e., the join of the itemsets ({a},{b}) is {1, 5, 6} will be used to represent the bipartite graph as follows: E={({a, b}↔{1, 5, 6}),({a, c}↔{1, 6}), ({a, d}↔{1}), ({a, e}↔{1, 6}), ({b, c}↔{1, 2, 3, 4, 6}), ({b, d}↔{1, 2, 4}),({b, e}↔{1, 2, 3, 6}), ({c, d}↔{1, 2, 4}), ({c, e}↔{1, 2, 3, 6}), ({d, e}↔{1, 2}). The fun-out of {a, b} is three due to the availability of the links directed to the set of transactions {1, 5, 6} and this value is greater than 2, hence {a, b} is regarded as a frequent 2-itemset, but the fun-out of {a, d} is one and this value is less than 2, hence {a, d} is regarded as an infrequent itemset. In the same manner, the construction of PBGs and support counting is accomplished for the rest of the items which results in F_2=({a, b},{a, c},{a, e},{b, c},{b, d},{b, e},{c, d},{c, e},{d, e}) are frequent itemset. These itemsets have 3, 2, 2, 5, 3, 4, 3, 4, and 2 support counts respectively.

3.3 K-itemset construct algorithm

The robustness and efficiency of LCOFI appear in this stage of FIs mining which depends on the properties of LCOA. The XORing, ORing, and ANDing are the main operations in LCOFI. After generating frequent 2-itemsets from frequent 1-itemsets. The next step includes generating frequent k-itemsets depending on the contents of the F_2 list

which contains all frequent 2-itemsets. The input of Algorithm 4 is a set of frequent (F_{k-1}) itemsets, while the outputs are a set of frequent (k-itemset). Steps #5 to #7 are in the same manner as the previous steps described in Algorithm 3, which will be used in the process to generate a frequent k-itemset. In step#10 the equality inclusion operation of LCOA will be used for this purpose as illustrated in Fig. 7. In step# 11 the algorithm selects two sets of frequent (k-1)-itemsets that have the equality inclusion property in such a way that they contain the same items except one item is different, for example, suppose that {a, b, c},{a, b, d} are two sets of frequent 3-itemsets that the items {a},{b} of two sets is the same itemsets, but items {c} and {d} are one different them, hence, the combination of two itemsets is {a, b, c, d}. Additionally, if the two PBGs have the *equality inclusion property* shown in step#12. Step# 14 includes combining the items represented as a bipartite graph by the OR operation template, i.e., (f_i-PBG OR f_j-PBG) to generate k-itemsets. Step# 15 make the AND operation of two tidlists of the combined frequent (F_{k-1}-itemset) from the OR operation to find the tidlist of the (k-items). Step#16 the support counting of the k-itemset is computed by fun-out of the item template assigned it and checked according to the min_sup threshold. In step#17 the frequent k-itemsets will be added to F_k, the list for frequent k-itemsets.These steps are repeated to find the rest frequent k-itemset, where k=3. Step#20 increases k by one to generate the next level of frequent itemsets. The frequent itemsets will be added to the set of frequent itemsets which are returned to the caller.

Algorithm 4. Construct k-itemsets(F_{k-1})

2. Input: F_{k-1}
3. Output: frequent k-itemsets
4. {
5. While($F_{k-1}\neq\{\}$) do
6. {
7. Remove f_i from (F_{k-1})
8. For (j=1 to length (F_{k-1}); j++) do
9. {
10. //*inclusion function of LCOA- Equality inclusion*
11.Boolean inclusionResult= inclusion(f-PBG, f_j-PBG);
12. If (inclusionResult)
13. {
14. f-PBG_{item}= f_i-PBG_{item} OR f_j-PBG_{item}; // ORing the items templates
15. f-PBG=anding(f-PBG);
16. if (fun-out (f-PBG) >=min-sup) // Support counting
17. F_k= $F_k\cup$f -PBG;
18. } if
19. }//for
20. k=k + 1;
21. } while
22. }// k-itemsets

For more explanation, consider Fig. 10. consider the following frequent 2-itemsets, F_2 list generated from the previous step which are F_2=({a, b}, {a, c}, {a, e}, {b, c},{b, d}, {b, e}, {c, d}, {c, e}, {d, e}). To give a full explanation, Recall example 1 and consider Figure 10 which includes steps to construct PBGs for two sets of frequent 2- itemsets as a sub bipartite graph from the F_2 list presented in section 3.2.

To generate frequent 3- itemset depending on the ORing operation the template ANDing operation, and the quality inclusion operation proposed for PBGs. Let's trace the process to generate a frequent 3-itemset shown in Fig. 10. Pick {a, b} as the first item from F_2 with each item of ({a, c}, {a, e}, {b, c}, {b, d}, {b, e}, {c, d}, {c, e}, {d, e})}. The equality inclusion operation will be applied to two PBGs from F2 in this process. The sets {a, b} and {a, c} have the equality inclusion property, which includes only one item different is {b} and {c}. This process can be described simply as the combination of items will be used to represent the bipartite graph as follows: E={({a, b, c}↔{1, 6}),({a, b, e}↔{1, 6}), ({a, c, e}↔{1, 6}), ({b, c, d}↔{1, 2, 4}), ({b, c, e}↔{1, 2, 3, 6}), ({b, d, e}↔{1, 2}), ({c, d, e}↔{1, 2})}. The fun-out of {a, b, c} is 2 due to the availability of the two itemsets link directed to two of transactions 1 and 6. This value is equal to 2, hence {a, b, c} is regarded as a frequent 3-itemset. In the same manner, the construction of PBGs and support counting is accomplished for the rest of the items which results in F_3= ({a, b, c}, {a, b, e}, {a, c, e}, {b, c, d}, {b, c, e}, {b, d, e}, {c, d, e}) are frequent itemset. These items have 2, 2, 2, 3, 4, 2, and 2 support counts respectively.

For the transaction database mentioned in Table 1 contains a frequent 4- itemset according to the min_sup threshold. Consider Fi. 5which shows the process generates a frequent 4-itemset depending on the contents of the F_3 list is presented in the previous step.

To generate a frequent 4-itemset, trace the same steps to generate a frequent 3-itemsets as shown in Algorithm 4. The {a, b, c },{a, b, e} are two sets of frequent 3- itemsets generated from the previous step. These sets include only one item different is {c} and {e}. This process can be described simply as the combination of items that will be used to

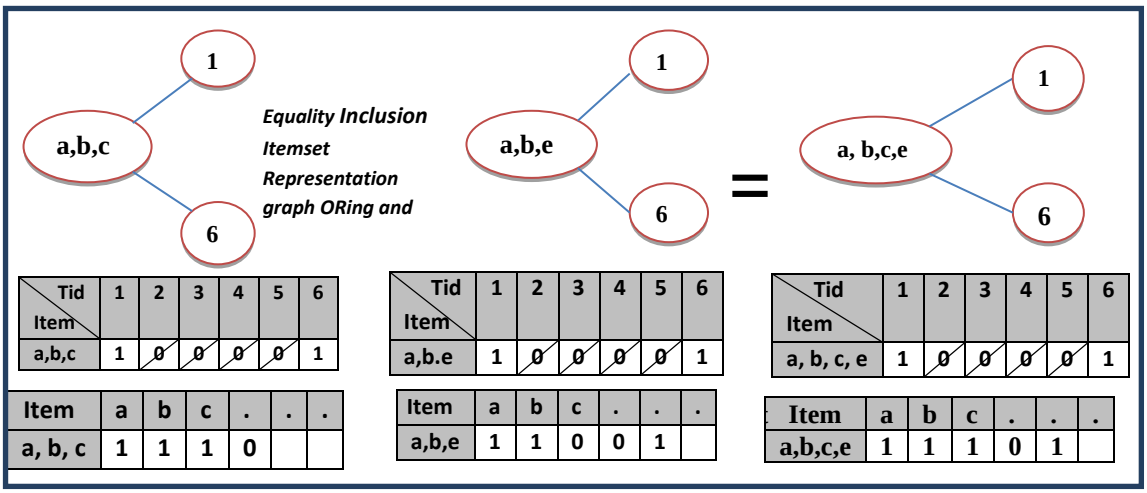


Figure. 11 ORing {a, b, c} and {a, b, e} item-template and their PBGs ANDing

represent the bipartite graph as follows: E={({a, b, c, e}↔{1, 6}), ({b, c, d, e}↔{1, 2})}. The fun-out of {a, b, c, e} and {b, c, d, e} are two due to the availability of the two itemsets link directed to the set of transactions, and the support of two itemsets is 2, hence {a, b, c, e} and {b, c, d, e} are regarded as frequent 4-itemset. For example, 1 mentioned in section 3.1 the final result is a set of all frequent itemsets generated in database D as output are ({a}, {b}, {c}, {d}, {e}, {a, b}, {a, c}, {a, e},{b, c}, {b, d},{b, e}, {c, d}, {c, e}, {d, e}, {a, b, c}, {a, b, e}, {a, c, e}, {b, c, d}, {b, c, e}, {b, d, e}, {c, d, e}, {a, b, c, e}, {b, c, d, e}).

4. Experiments datasets

Experiments were carried out on an Intel(R) Core(TM) i5-5300U CPU running at 2.30 GHz, 8 GB of memory, and a PC running Windows 10. The algorithm was implemented using Python. The experiments indirectly pointed to memory utilization according to the used dataset and min_sup values. It is well known that the lower value of min_sup leads to the largest number of FIs, in addition to the increment in execution time and memory consumption. The experiments are executed by using four typical datasets, such as chess, mushrooms, T10I4D100K, and T40I10D100K. To better test and analyze the performance of the algorithm, four different types of public data sets are used, which have obvious differences in the amount of data, transaction width, number of items, sparseness, and so on. The data sets used in the experiments are of two types; real and synthetic public test datasets commonly used in association algorithm research, which are publicly available in the frequent itemset mining implementations repository [34]. The characteristics of this dataset are shown in Table 5.

5. Results and discussion

This section deals with the results related to LCOFI and also comparative analyzes of the results with that of the Apriori and FP-Growth algorithms using different four experiments to generate frequent itemsets depending on execution time and memory usage. The results presented in this paper were obtained from the Python authors' implementation of the apriori and FP-growth algorithms and are fully consistent with the results in the references [35-38] related to these algorithms. The results are presented in tables as follows:

Table 6 presents the number of frequent itemsets hidden in the chess dataset using min_sup of 40%, 50%, 60%, 70%, 80%, and 90%, which are 979582, 32273, 28173, 24401, 196, and 13 respectively. These FIs numbers are for the Apriori, Fp-growth, and LCOFI. If the min_sup value is less than 40%, then a huge number of frequent itemsets will be generated, increasing the execution time of these algorithms and memory consumption. It is clear that the number of frequent itemsets increases dramatically when the value of the min_sup threshold decreases, i.e., the lower value of min_sup leads to the largest number of FIs.

Table 7 presented the total of frequent itemsets using mushroom dataset and different min_sup values. The results in Table 7 explain the total of frequent itemsets of mushroom dataset under min_sup of 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and 100% is 342869, 12248, 1867, 304, 17, 9, 5, 5, 4, and 1 respectively.

Using the T10I4D100K dataset and various min sup values ranging from 1% to 9% of the dataset size, Table 8 shows the total number of frequent

Table 5. Dataset characteristics

Dataset Name	No.of Items	No. of Transactions	Dataset Type	Size(KB)
Mushroom	120	8124	Real /very dense	335
Chess	76	3196	Real/very dense	558
T10I4D100K	1000	100000	Artificial/sparse	5,774
T40I10D100K	1000	100000	Artificial/sparse	18,768

Table 6. Total of frequent itemset with different min_sup values under the chess data set

Min_sup / Dataset	40%	50%	60%	70%	80%	90%
Chess	979582	32273	28173	24211	196	13

Table 7. Total of frequent itemset with different min_sup values under the mushroom data set

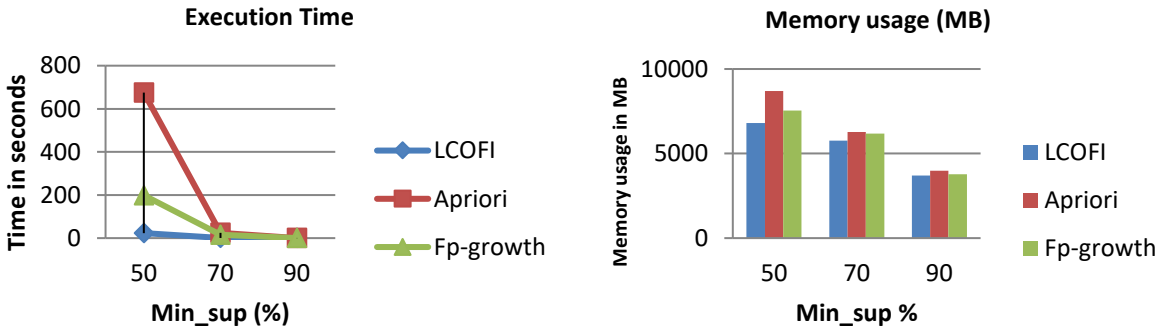
Min_sup / Dataset	10%	20%	30%	40%	50%	60%	70%	80%	90%	100 %
Mushroom	342869	12248	1867	304	17	9	5	5	4	1

Table 8. Total of frequent itemsets with different min_sup values under the T10I4D100K dataset

Min_sup / Dataset	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
T10I4D100K	384	155	60	26	10	4	2	0	0	0

Table 9. Total of frequent itemsets with different min_sup values under the T40I10D100K dataset

Min_sup / Dataset	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
T40I10D100K	24732	2273	780	430	315	283	183	137	110	82

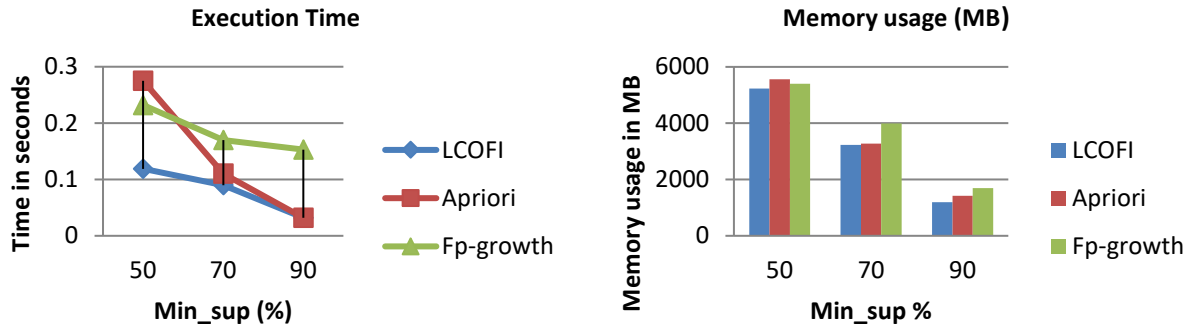


(a) (b)

Figure. 12 Shows a comparative graph of three algorithms such that the line graph: (a) execution time using the chess dataset, while the bar graph and (b) memory usage using the chess dataset

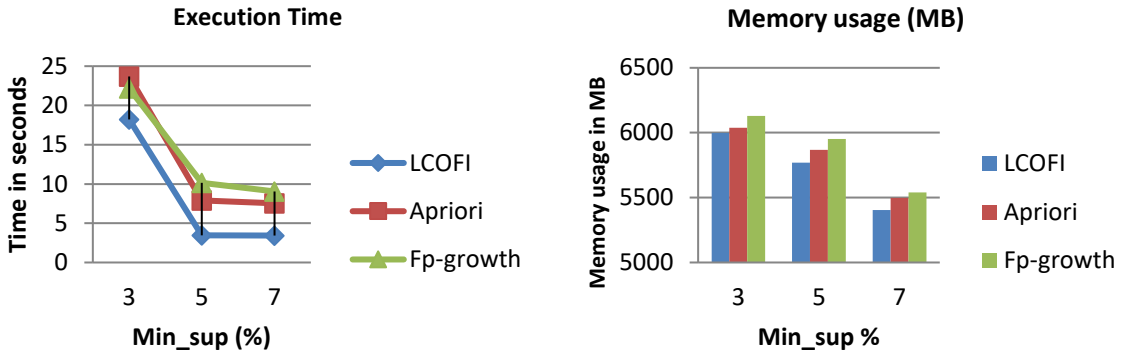
itemsets. The findings in Table 8 give 384, 155, 60, 26, 10, 4, 2, 0,0, and 0 as the total number of frequent itemsets of the T10I4D100K dataset under min sup of 1%, 2%, 3%, 4%, 5%, 6%, 7%, 8%, 9%, and 10%, respectively. Itemsets are not available when the min sup is larger than or equal to 8% since there are no FIs with a high support value in this sparse dataset. The presence of rare items, as well as the fact that the shortest transaction is 4 items long and the average transaction length is 37, could be attributed to this phenomenon.

The results in Table 9 explain the total of frequent itemsets of T40I10D100K dataset undermin_sup 1%, 2%, 3%, 4%, 5%, 6%, 7%,



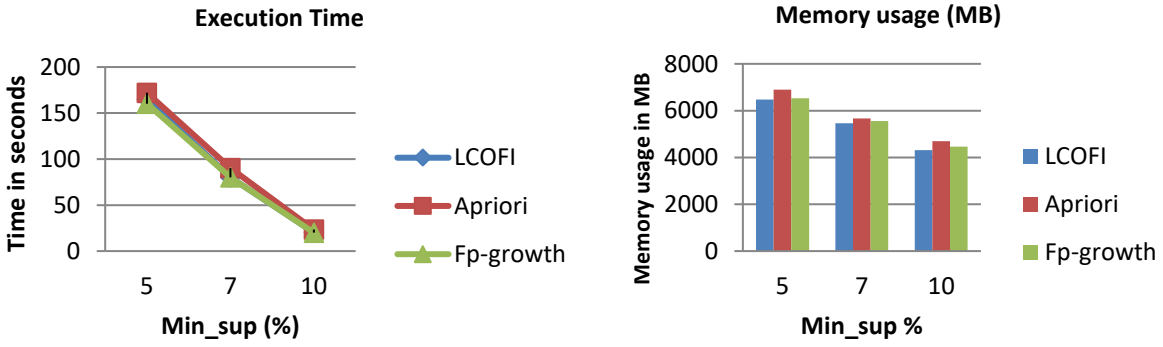
(a) (b)

Figure. 13 A comparative graph of three algorithms such that the line graph: (a) execution time using mushroom dataset, while the bar graph and (b) memory usage on mushroom dataset



(a) (b)

Figure. 14 A comparative graph of three algorithms such that the line graph: (a) execution time on T10I4D100K dataset, while the bar graph and (b) memory usage on T10I4D100K dataset



(a) (b)

Figure. 15 A comparative graph of three algorithms such that the line graph: (a) execution time on T40I10D100K dataset, while the bar graph and (b) memory usage on T40I10D100K dataset

8%,9% and10% are 24732, 2273, 780, 430, 315, 283,183, 137, 110, and 82 respectively. This dataset is more sparse than mushroom in addition to the fact that it contains more than 50% of the items with a very low number of occurrences.

Fig. 12(a) shows a comparative line graph of execution time for the Apriori, Fp-growth, and LCOFI algorithms on the chess dataset. The algorithms' parameters are various min_sup values and the chess dataset. LCOFI outperforms the Apriori algorithm and the Fp-growth algorithm when the min_sup value is 50%, 70%, or 90%, but the Fp-growth algorithm is better in terms of execution time than the Apriori algorithm. Fig. 12(b) shows a comparative bar graph of memory usage for three algorithms on chess dataset. The

proposed algorithm, LCOFI consumes less memory to generate the frequent itemsets compared with the Apriori algorithm and the Fp-growth algorithm under the min_sup value of 50%, 70%, and 90%, where the Fp-growth consumes less memory than the Apriori algorithm under various min_sup. In the same manner, we present the comparisons of execution time and memory consumption among the three algorithms using the adopted databases. Figs. 12-15 show how LCOFI outperforms in most experiments with different values of min-sup and these databases.

6. Conclusion

In this paper, a new algorithm for frequent itemset mining is proposed that takes advantage of the LCO algorithm's characteristics and the operations proposed in it to deal with the bipartite graph, which is used to minimize the logic expressions and circuits. The algorithm proved that the similarity between the binary representation of transactional databases and the truth table of a logical problem and the similarity between the PBGs and the suggested binary representation of itemsets led to an efficient algorithm for mining FIs depending on bipartite graph representation and its operations: XORing, ORing, ANDing, and the inclusion property of LCOA. The experiment results show that LCOFI reduces the consumption of time and memory with various min_sup values on different datasets compared with Apriori and Fp-growth algorithms. The outperformance is gained from many properties involved in LCOFI, such as:

- It required one database scan operation,
- It is not necessary to perform the operations of prying the candidate itemsets apart.
- Complex data structures, such as hash tables or long linked lists, are unnecessary.
- There is no need to apply complex operations to joint k-itemsets to generate (k+1)-itemsets.
- There is no need to perform a sorting operation to keep items in an itemset in chronological order.
- The counting of itemsets' supports became no more than the counting of the number of 1s in a binary vector.

Different datasets were used to test the LCOFI. These datasets have various characteristics, such as the number of transactions, the average size of transactions, the number of items, the density, and the type, such as real or synthetic. These tests were done to verify the LCOFI's scalability and tolerance for the sparseness and length of the generated itemsets.

As a future step, one can plan for LCOFI modification to mine frequent patterns in different application databases.

Conflicts of interest

The authors declare no conflict of interest.

Author contributions

The concept, method, formal analysis, validation, references, data collection and original draft preparation were all contributed by Oday Ahmed Al-Ghanimi and Hussein K. Khafaji; the superviser. Hussein K. Khafaji revised and edited the work. Algorithms' implementation is accomplished and obtaining the results are done by Oday Ahmed Al-Ghanimi.

References

[1] N. F. Zulkurnain and A. Shah, “HYBRID: An efficient unifying process to mine frequent itemsets”, In: *Proc. of 2017 IEEE 3rd International Conference on Engineering Technologies and Social Sciences (ICETSS)*, pp. 1–5, 2017, DOI: 10.1109/ICETSS.2017.8324140.

[2] M. A. Shouman, M. Aziz, A. H. N. Ahmed, and N. Z. A. Fatah, “A Comparative Study of SQL Based Approaches for Mining Association Rules over Relational DBMS”, *Egypt. Comput. Sci. J.*, Vol. 34, No. 2, 2010.

[3] M. Al-Maolegi and B. Arkok, “An improved Apriori algorithm for association rules”, *ArXiv Prepr. arXiv1403.3948*, 2014.

[4] A. Dhabi, S. Jabri, Y. Balouki, and T. Gadi, “A new method to select the interesting association rules with multiple criteria”, *Int. J. Intell. Eng. Syst.*, Vol. 10, No. 5, pp. 191–200, 2017, doi: 10.22266/ijies2017.1031.21.

[5] W. A. Salman and S. B. Sadkhan, “Status and Challenges of Frequent Itemsets and Association Rules Mining Methods”, In: *Proc. of 2020 3rd International Conference on Engineering Technology and its Applications (IICETA)*, 2020, pp. 154–158, DOI:10.1109/IICETA50496.2020.9318834.

[6] S. Singh and J. Singh, “Association rules and mining frequent itemsets using algorithms”, *Int. J. Comput. Sci. Eng. Technol*, Vol. 3, pp. 370–373, 2012.

[7] W. X. gang, “A Summary of Research on

Frequent Itemsets Mining Technology", *Procedia Comput. Sci.*, Vol. 131, pp. 841–846, 2018, DOI: https://doi.org/10.1016/j.procs.2018.04.276.

[8] C. H. Chee, J. Jaafar, and I. A. Aziz, "FP-NoSQL: An Efficient Frequent Itemset Mining Algorithm Using the FP-DB Approach", In: *Proc. of 2018 IEEE Conference on Big Data and Analytics (ICBDA)*, 2018, pp. 80–86.

[9] R. Agrawal, T. Imielinski, and A. N. Swami, "Association rules between sets of items in large databases", In: *Proc. of ACM SIGMOD Int. Conf. on Management of Data, Washington*, 1993, pp. 207–216.

[10] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo, "Fast discovery of association rules", *Adv. Knowl. Discov. data Min.*, Vol. 12, No. 1, pp. 307–328, 1996.

[11] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, and T. Yiu, "MAFIA: A maximal frequent itemset algorithm", *IEEE Trans. Knowl. Data Eng.*, Vol. 17, No. 11, pp. 1490–1504, 2005, DOI: 10.1109/TKDE.2005.183.

[12] M. J. Zaki and C. J. Hsiao, "CHARM: An efficient algorithm for closed itemset mining", In: *Proc of the 2002 SIAM International Conference on Data Mining*, 2002, pp. 457–473.

[13] M. J. Zaki and C. J. Hsiao, "Efficient algorithms for mining closed itemsets and their lattice structure", *IEEE Trans. Knowl. Data Eng.*, Vol. 17, No. 4, pp. 462–478, 2005, DOI: 10.1109/TKDE.2005.60.

[14] V. Tiwari, V. Tiwari, S. Gupta, and R. Tiwari, "Association rule mining: A graph-based approach for mining frequent itemsets", In: *Proc. of ICNIT 2010 - 2010 International Conference on Networking and Information Technology*, 2010, pp. 309–313, DOI: 10.1109/ICNIT.2010.5508505.

[15] D. S. Kumar, N. Srinivasu, and S. S. R. P. Ch, "Bipartite graph based frequent pattern mining using maximum degree of a vertex", *Bulletin Monumental*, Vol. 22, No. 7, 2021.

[16] M. J. Islam, M. G. Hussain, B. Sultana, M. Rahman, M. S. Rahman, and M. A. Rahaman, "Simplifying the Boolean Equation Based on Simulation System using Karnaugh Mapping Tool in Digital Circuit Design", *GUB J. Sci. Eng.*, pp. 76–84, 2020.

[17] H. G. Vu, N. D. Bui, and A. T. Nguyen, "Performance Evaluation of Quine-McCluskey Method on Multi-core CPU", In: *Proc. of 2021 8th NAFOSTED Conference on Information and Computer Science (NICS)*, 2021, pp. 60–64.

[18] M. Bolton, *Digital systems design with programmable logic*, Addison-Wesley Longman Publishing Co., Inc., 1990.

[19] L. E. Frenzel Jr., *Practical Electronic Design for Experimenters*, McGraw-Hill Education, 2020.

[20] N. J. Glauch, D. S. Choi, and G. Herman, "How engineering students use domain knowledge when problem‐solving using different visual representations", *J. Eng. Educ.*, Vol. 109, No. 3, pp. 443–469, 2020.

[21] N. Sharma and A. Singh, "K-partition model for mining frequent patterns in large databases", *Int. J. Comput. Sci. Eng.*, Vol. 4, No. 9, p. 1505, 2012.

[22] C. W. Kann, "Digital Circuit Projects: An Overview of Digital Circuits Through Implementing Integrated Circuits", *Getysburg College Open Educational Resources*, 2014.

[23] W. V Quine, "A way to simplify truth functions", *Am. Math. Mon.*, Vol. 62, No. 9, pp. 627–631, 1955.

[24] E. D. Nugroho, "Development of Applications for Simplification of Boolean Functions using Quine-McCluskey Method", *Telematika*, Vol. 18, No. 1, p. 27, 2021, doi: 10.31315/telematika.v18i1.3195.

[25] A. Y. Khedr, R. A. Ramadan, and S. M. A. Magid, "QMR: QUINE-MCCLUSKEY for rule minimization in rule-based systems", *Int. J. Intell. Comput. Inf. Sci.*, 2012.

[26] R. K. Brayton, G. D. Hachtel, C. McMullen, and A. S. Vincentelli, "Logic minimization algorithms for VLSI synthesis", *Springer Science & Business Media*, Vol. 2, 1984.

[27] E. F. Ashmouni, R. A. Ramadan, and A. A. Rashed, "Espresso for Rule Mining", *Procedia Comput. Sci.*, Vol. 32, pp. 596–603, 2014.

[28] O. A. A. Ghanim and H. K. Khafaji, "A new logic circuits optimization algorithm using a bipartite graph", *Indonesian J Elec Eng & Comp Sci*, Vol. 99, No. 1, Month 2099: 1-1x, Vol. 28, No. 3, pp. 1621–1632, 2022, DOI: 10.11591/ijeecs.

[29] Y. Rochd and I. Hafidi, "An Efficient Distributed Frequent Itemset Mining Algorithm Based on Spark for Big Data", *International Journal of Intelligent Engineering and Systems*, Vol. 12, No. 4, 2019, DOI: 10.22266/ijies2019.0831.34.

[30] S. Rubaiee, M. Masud, R. Alroobaea, G. S. Gaba, Z. Jian, and Z. Zhao, "An improved association rule mining algorithm for large data", *J. Intell. Syst.*, Vol. 30, No. 1, 2021.

[31] A. Frieze and M. Karoński, *Introduction to Random Graphs*, Cambridge University Press,

2016.
[32] A. S. Asratian, T. M. J. Denley, and R. Häggkvist, *Bipartite Graphs and Their Applications*, Vol. 131, Cambridge university press, 1998.
[33] J. V. D. Brand, "Bipartite matching in nearly-linear time on moderately dense graphs", In: *Proc. of 2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 919–930, 2020.
[34] "Frequent Itemset Mining Dataset Repository", http://fimi.uantwerpen.be/data/.
[35] Y. Zhang, W. Yu, Q. Zhu, X. Ma, and H. Ogura, "Right ‐ hand side expanding algorithm for maximal frequent itemset mining", *Appl. Sci.*, Vol. 11, No. 21, pp. 1–18, 2021, doi: 10.3390/app112110399.
[36] L. Xu, "Improvement and Application of Apriori Algorithm Based on Equalization", In: *Proc. of 2019 IEEE Fourth International Conference on Data Science in Cyberspace (DSC)*, pp. 635–641, 2019, doi: 10.1109/DSC.2019.00104.
[37] C. Wu and H. Jiang, "Research on Parallelization of Frequent Itemsets Mining Algorithm", In: *Proc. of 2021 IEEE 6th Int. Conf. Cloud Comput. Big Data Anal. ICCCBDA 2021*, No. 2018, pp. 210–215, 2021, doi: 10.1109/ICCCBDA51879.2021.9442547.
[38] M. M. Hasan and S. Z. Mishu, "An Adaptive Method for Mining Frequent Itemsets Based on Apriori And FP Growth Algorithm", In: *Proc. of 2018 International Conference on Computer, Communication, Chemical, Material and Electronic Engineering (IC4ME2)*, pp. 1–4, 2018, doi: 10.1109/IC4ME2.2018.8465499.