A new logic circuits optimization algorithm using bipartite graph

Oday Ahmed Al-Ghanimi[1], Hussein K. Khafaji[2]
[1]Department of Computer Science, Informatics Institute for Postgraduate Studies, Baghdad, Iraq
[2]Department of Communication Engineering, Al-Rafidain University College, Baghdad, Iraq

Article Info

Article history:

Received Jun 10, 2022
Revised Aug 19, 2022
Accepted Sep 2, 2022

Keywords:

Bipartite graph
Espresso
Karnaugh map
Logic circuit optimization
Logic expression

ABSTRACT

Designing a logic circuit from the scratch requires its description in logical expression, (e.g. sum of products), and then the expression should be optimized to diminish the cost and complexity of the circuit by reducing the number of literals, the number of logical terms, and/or logical operations. Karnaugh map, K-Map, is the most popular method in the optimization process, but it suffers from many drawbacks such as its inefficiency or the inability to be used in minimizing logical expression containing more than four literals, in addition to the complexity of implementing it as a program. In this paper, we propose a new algorithm to optimize the logic circuits depending on the bipartite graph and some of the suggested mathematical operations. The proposed algorithm is simple for programming implementation, literal-unlimited number, and is easy to be visualized and understandable. Many of the logic circuits of 3, 4, 5, and 6 literals were optimized and the results were correctly matched with the results of the Karnaugh map. Also, tens of logic circuits of more than 6 literals are optimized and the results were correctly checked with their truth tables and Logic-Friday tool.

Corresponding Author:

Oday Ahmed Al-Ghanimi
Department of Computer Science, Informatics Institute for Postgraduate Studies
Baghdad, Iraq
Email: ms202030612@iips.icci.edu.iq

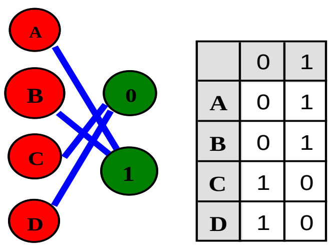
1. INTRODUCTION

The cost of logic circuits is growing according to its complexity resulted from the number of literals, number of terms, and number of logical operations that initially involved in their logical expressions. Therefore, the logic expression should be optimized as a pre-step of implementation of the circuits. The optimization here means how to transform the expression to the simplest but correspondent one. This process is not frank such that the connections of the expression/circuit components may form a challenge for engineers in the design and/or reengineering phases. Originally the description of the circuit and what should do is extracted from what so-called truth table. The design task is largely to determine what type of circuit will perform the function described in the truth table [1].

A truth table is a mathematical way of representing the logical relationship between the inputs, i.e. the influencing and active factors in a problem, and the logical influence of those factors, i.e. the outcomes. It shows the information of three entities; The Boolean function, which may be in the form of a Boolean expression, the inputs and their diversity of values, and the outputs that change according to the input probabilities [2]. In particular, truth tables can be utilized to present the truthiness or falseness of a propositional expression for all input value combinations, that is, logically valid. For a given problem of n binary input, the truth table will include 2n entries corresponding to possible combinations of the input variation values. A Boolean function results in true or false according to the input combination therefore the number of given function may produce true or false for each combination so the number of Boolean functions of n variables is the double exponential 2^{2^n} [3].

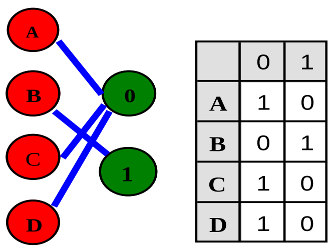
The Karnaugh map or K-map was invented to optimize the digital logic circuit depending on the truth table. K-map is not without flaws; it is difficult to be implemented as software and difficult to use according to many researches [4]. Also, it is very unclear when a problem contains more than four parameters. Four literals produce 24=16 combinations. The designer deals with the combinations that result in 1/true as a result of the circuit. The selection of these combinations and the configuration of the optimized circuit are tedious and error-prone [5]. Willard Quine and Edward McCluskey developed the first alternative method for K-Map which is known as the tabular method. It is beginning with a truth table and ends with a systematic procedure to determine the set of minimum prime implicants released by the output functions [6], [7]. Quine McCluskey algorithm lends itself to be automated as a computer program, but it is inefficient in terms of execution time and memory consumption such that adding an extra literal will almost double these two ramifications of the minimization cost [8]. In a conclusion, the Quine McCluskey algorithm is efficient for a restricted number of input literals and output functions in addition to its farness from understandability and visualization [9], [10]. Brayton *et al*. [11] developed what so-called ESPRESSO algorithm that keeps a very accepted level of computer resources usage and performance efficiency. ESPRESSO as a program iterates to manipulate "cubes" representing the product terms, therefore its minimized output is not assured to produce optimal minimization, in addition to its dependency on vectors optimization which makes it a loser to the characteristic of visualization and easiness of understanding [12]. Many tools for logic circuit minimization have been designed most of these tools depend on ESPRESSO algorithm [13]. One of these tools is Logic Friday. it is free under Windows software that grants a GUI to Espresso. The function and the input to Logic Friday can be in many forms such as gate diagrams, equations, or truth tables. In 2012, the newer update of Logic Friday was released in version 1.1.4. This paper presents a new method for optimizing digital circuits/logical expressions. It utilizes the bipartite graph properties and some of the suggested operations on the graphs to eliminate the constraints of truth tables and the K-Map approach. Therefore, the next part of the introduction is related to the 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; N0 and N1. Every edge e in the edge set E links a node in N0 to one node in N1 [14], [15]. The node sets N0={n01, ... n0n} and N1={n11 ... n1m} are the mutually exclusive vertices sets and they are called graph's parts [14], [16]. $E \subset N0 \times N1$ is a set of edges that connect vertices between two partitions [16], [17]. Figure 1 show two examples of bipartite graphs with their biadjacency matrices. The graph includes the edges (A, 1), (B, 1), (C, 0), and (D, 0) has been illustrated in Figure 1(a) and the graph involves the edges (A, 0), (B, 1), (C, 0), and (D, 0) has been illustrated in Figure 1(b).



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

(a)



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

(b)

Figure 1. An example of bipartite graphs with their biadjacency matrices such that the graph in (a) includes the edges (A,1), (B, 1), (C,0), and (D, 0), while (b) involves the edges (A,0), (B, 1), (C,0), and (D, 0).

The discrepancy between the size of N_1 and N_2 and the direction of the relationships, edges, between the nodes of the two sets only, approximate the execution time of matching and traversing a bipartite graph closer to the linear time in most circumstances [17]. There are unlimited number of applications for bipartite graph such as search engines, social networks [18], and recommendation systems [19], data and networks classification [20], [21], cloud computing [22], health care, biology and medicine [23]. Other applications relevant to bipartite graphs are related to x-ray crystallography, metabolic pathways, chemical reaction networks, missile guidance, routing and wavelength assignment problem, and metabolic pathways [24], [25], and the list of applications is rapidly growing. In this paper, a novel method is presented for logic circuit/expression optimization which mainly depends on bipartite graph.

2. THE PROPOSED METHOD FOR LOGIC CIRCUIT OPTIMIZATION (LCOA)

The proposed method depends mainly on using bipartite graphs to represent entries of the truth table. It applies suggested operations on biadjacency matrices of these graphs. So, we will introduce the concept of product bipartite graph (PBG) to simplify the understanding of the proposed method**.**

2.1. Product bipartite graph

Each literal in the logical problem can have one of two values true/1 or false/0. A truth table presents all combinations of the literals that lead to 1 or 0 as an output for the digital circuit or logical expression. The logical expression can be constructed by the sum of products (SOP), or product of sum (POS). Initially in this paper, we'll concentrate on SOP in which only the combinations that lead to logic 1 output will be considered. Consider the truth table presented in Table 1 and the logical expression obtained from it using SOP.

Table 1. Truth table

Seq.	A	B	C	D	Output
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	**0**	**1**	**0**	**0**	**1**
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	**1**	**1**	**0**	**0**	**1**
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

$$Output = A'\,B'\,C'\,D' + A'\,B'\,C'\,D + \mathbf{A'\,B\,C'\,D'} + A'\,B\,C'\,D + A'BCD' + \mathbf{ABC'D'}$$

In this paper, we introduce the concept of PBG. PBG is a bipartite graph with two sets of nodes; N_1 is the set of literals of the logical problem and N_2 consists of binary values 0 and 1. Recall Figure 1 includes the biadjacency matrix of the PBG. Also, Figure 1(a), represents the PBG of the combination ABC'D' which corresponds to 1, 1, 0, and 0 as values for the literal A, B, C, and D respectively. This PBG represents the 12th entry of the truth table, i.e., 1100.

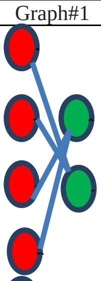
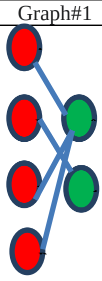
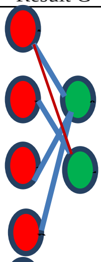
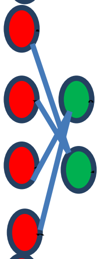
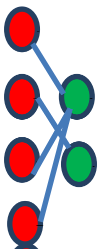
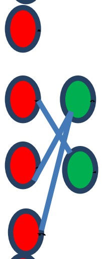
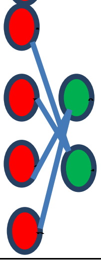
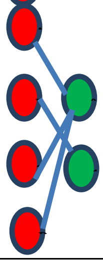
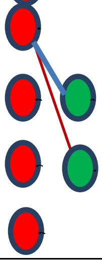
2.2. Suggested operations on PBG

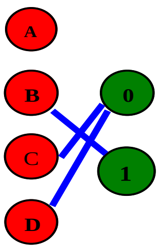
The ORing operation of two bipartite graph results in a graph with all the edges of the two graphs. The ANDing operation keeps the common edges of the two bipartite graph. XORing keeps the different edges of the two bipartite graphs in the resultant bipartite graph, (see Figure 2).

Table 2 summarizes the suggested operations depending on the PBGs presented in Figure 1(a) and 1(b). Figure 1(b) represents another PBG for another entry of the truth table presented in Table 1. This entry is 0110 i.e., the 6th entry A'BCD'.

From Table 2, it is possible to conclude that the ORing operation of two bipartite graphs results in a bipartite graph that represents their union. The ANDing operation of two bipartite graphs results in an intersection bipartite graph. The XORing operation of two bipartite graphs keeps the different edges of the two bipartite graphs. These suggested operations can be utilized in many applications in addition to their utilization to achieve the goal of this research. Some modifications will be done to reach the aim of the research. These modifications are done because the resultant graphs may contain an entry that indicates that a literal is connected with node 0 and node 1 and this case is logically impossible in this research context, but it is very useful because it indicates important information related to the minimization process. For example, the output related to literal A of XOR operation in Table 2 is [1,1]. This indicates that the PBG1 includes A’ and PBG2 includes A or vice versa. Let’s call this case, [1, 1], conflict edge. When we eliminate the conflict edge of the XOR operation of Table 2 from PBG1 or PBG2, we obtain the bipartite graph presented in Figure 2.

Table 2. Summarization of the proposed operations on the PBGs

Seq.	PBG1	Graph#1	Op.	PBG2	Graph#1	Result PBG	Result G	Comment
1	0 1 A 0 1 B 0 1 C 1 0 D 1 0		OR	0 1 A 1 0 B 0 1 C 1 0 D 1 0		0 1 A 1 1 B 0 1 C 1 0 D 1 0		Union of The graphs' Edges
2	0 1 A 0 1 B 0 1 C 1 0 D 1 0		AND	0 1 A 1 0 B 0 1 C 1 0 D 1 0		0 1 A 0 0 B 0 1 C 1 0 D 1 0		Intersection of the graphs' edges
3	0 1 A 0 1 B 0 1 C 1 0 D 1 0		XOR	0 1 A 1 0 B 0 1 C 1 0 D 1 0		0 1 A 1 1 B 0 0 C 0 0 D 0 0		Difference of the graphs' edges

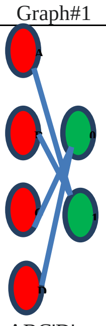
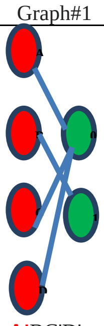
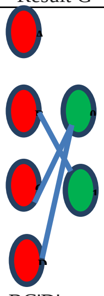
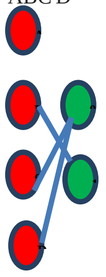
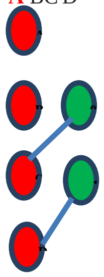
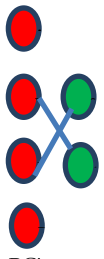


	0	1
A	0	0
B	0	1
C	1	0
D	1	0

Figure 2. Elimination of different edge of two PBGs

This process is similar to the elimination of unstable literal in K-map. Its leads to the minimization of literals and/or logical operators that control the circuit complexity. The XOR operation will play a crucial role in the proposed method of digital circuit optimization because it depends mainly on the gradual elimination of different edges in the PBGs. Now, Let's define a property named "*inclusion*" for two bipartite graphs which has two cases; named "*equality-inclusion*" and "*partial-inclusion*". Table 3 summarizes the inclusion property. We define the "*Equality-inclusion*" case, for the purposes of this research, as the case when the two bipartite graphs have the same number of vertices and the same number of edges such that the vertices that are having edges are the same in each one but there exists one different edge between them. The "*partial-inclusion*" is defined as the case when the two bipartite graphs have the same number of vertices and the difference between the numbers of edges equals 1 with one edge differs.

Table 3. Summarization of the inclusion property of the PBGs

Seq.	PBG1	Graph#1	Op.	PBG2	Graph#1	Result PBG	Result G	Comment
1	$\begin{array}{ccc} & 0 & 1 \\ A & 0 & 1 \\ B & 0 & 1 \\ C & 1 & 0 \\ D & 1 & 0 \end{array}$ ~~A~~BC'D'	 ABC'D'	(EQUALITY) and AND	$\begin{array}{ccc} & 0 & 1 \\ A & 1 & 0 \\ B & 0 & 1 \\ C & 1 & 0 \\ D & 1 & 0 \end{array}$ A'BC'D'	 ~~A'~~BC'D'	$\begin{array}{ccc} & 0 & 1 \\ A & 0 & 0 \\ B & 0 & 1 \\ C & 1 & 0 \\ D & 1 & 0 \end{array}$ BC'D'	 BC'D'	The "equality-inclusion" property and "AND" operation will be used for logic circuit optimization. The result of the optimization is BC'D' with ignoring PBG1 and PBG2.
2	$\begin{array}{ccc} & 0 & 1 \\ A & 0 & 0 \\ B & 0 & 1 \\ C & 1 & 0 \\ D & 1 & 0 \end{array}$ BC'~~D'~~	 BC'D'	(PARTIAL) and AND	$\begin{array}{ccc} & 0 & 1 \\ A & 0 & 0 \\ B & 0 & 0 \\ C & 1 & 0 \\ D & 0 & 1 \end{array}$	 C'D	$\begin{array}{ccc} & 0 & 1 \\ A & 0 & 0 \\ B & 0 & 1 \\ C & 1 & 0 \\ D & 0 & 0 \end{array}$ BC'	 BC'	The "partial-inclusion" property and "AND" operation will be used for the logic circuit to optimize the larger PBG. **The result is BC' + C'D**

2.3. The algorithms of the proposed method

The proposed digital circuits optimization Algorithm consists of the following general steps and its pseudo code is presented in Algorithm 1:

Algorithm 1. Logic circuit optimization

```
Algorithm, LCOA

Global variable biadjacency matrix B1, B2;
Global variable B1_literals , B2_literals;
Algorithm: Logic Circuit Optimization
Input:
     Truth table T;
Output:
        Optimized Logic Circuit OLC; // OLC is global var.
 {               OLC= { };
        Construct the SOP terms list L;
        //it is better to implement L as a queue.
        Construct the PBG of each term l ∈ L;
        //i.e. construct the biadjacency matrix
        While (L<>{} ) do
        { If (picking_ two_One-edge-Differes(L)==true)
            // Fetch 2 bipartite graphs B1 and B2 from L
            // having inclusion property.
         {
               optimized_Graph= anding (B1, B2);
               //add optimized_Graph to L
               enqueue (optimized_Graph, L);
               // if (length(B1_literals)== length(B2_literals))
               // do nothing; just
               // ignore B1 and B2; they are already de-queued
               if (length(B1_literals) > length(B2_literals))
                  enqueue (B2, L); //and ignore B1
               else if (length(B1_literals) < length(B2_literals))
                   enqueue (B1, L); //and ignore B2
           } // if
             } // while
             return OLC;
} // OLC
```

– Depending on the truth table or the number of true combinations construct the SOP expression.
– Construct the biadjacency matrices of each PBG in the SOP expression.
– 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, $(|E(PBG_i)|\leq|E(PBG_j)|)$, (consider the inclusion algorithm-

Algorithm 2 and recall inclusion definition in section 3.1, and they have only one different edge, consider XORing algorithm, Algorithm 3. The process of elimination occurs in the larger PBG with keeping the resulting PBG and the smaller PBG, or that the elimination process occurs on both PBGs in the case of equality and keeping the resulting PBG and neglecting the main PBGs. The elimination process is accomplished by Algorithm 4, ANDing algorithm.

– Repeat step 3 until the SOP is optimized.

The pseudo-code of the algorithms of the proposed method is presented in Algorithm 1 which is written in a self-documented manner.

Algorithm 2. Inclusion algorithm
```
// Returns true if G1⊆ G2 or G2⊆ G1
boolean inclusion(G1, G2)
{
 G1_literals={}; G1_literals={}; c1=c2=0;
 for (i=1; i<= No. of literals; i++)
 {
   if (row_i of G1<>[0, 0])
      { G1_literals= G1_literals ∪ i);
        c1++;
     }
  if (row_i of G2<>[0, 0])
    { G2_literals= G2_literals ∪ i);
       c2++;
    }
  } // for
  if (G1_literals == G2_literals)
    return true;
  else if (absolute (c1-c2)==1) and
           (G1_literals ⊆G2_literals) or
           (G2_literals ⊆ G1_literals))
          return true;
        else return false;
}// inclusion
```

Algorithm 3. XORing algorithm
```
boolean xoring(G1, G2)
{
 int c=0;
 for (i=1, i<= No.of literals; i++)
  {
    v=(row_i of G1) xor (row_i of G2);
        if(v== [1, 1])
         if (++c>1) //more than 1 difference
        { enqueue(G2, L);return false;}
   } //for
 return true;
}
```

Algorithm 4. Graph-ANDing algorithm
```
biadjacency anding (G1, G2)
{
 biadjacency tempBG;
 for (i=1, i<= No. of literals; i++)
 if (row_i of G1= =[0, 0])
    //Optimized row
              row_i of tempBG = row_i of G2;
      else
             row_i of tempBG= (row_i of G2) and
                (row_i of G1);
  return tempBG;
} // anding function
```

Algorithm 5 Boolean picking_two_0ne-edge-Differes(L) is responsible for finding two PBGs having inclusion property.

Algorithm 5. Boolean picking_two_0ne-edge-Differes(L)

```
{
 dequeue(G1, L);
 dequeue(G2, L);
 tempG=G2;
 for (i=1; i<=|L|-2; i++)
 {
  if (inclusion(G1, G2)= =true)
    if(xoring (G1,G2)= =true)
      return true;
    else
    { enqueue(G2, L);//re-insert G2 in L
               dequeue(G2,L);//Fetch new G from L
               if (G2==tempG)
              { OLC=OLC ∪ G1;//G1 is optimized
               G1=G2;
      continue;
     }
   }//else
 }// for
 if (I >|L|-2) return false;
}
```

2.4. LCOA and K-Map of three literals

Consider the truth table presented in Table 4 and its SOP, O=a'b'c'+a'b'c+a'bc'+a'bc+ab'c+abc. Also, consider Table 5 and Table 6 which presented the optimization depending on K-map and LCOA respectively. Consider Table 6, LCOA initially fetches two PBGs; AB’C and ABC, then the proposed ANDing will be applied to obtain AC PBG. The AB’C and ABC PBGs will be neglected. Then LCOA will elite A’BC’ and A’BC PBGs to obtain A’B. Finally, it manipulates A’B’C’ and A’B’C PBGs to obtain A’B’ PBG. The turn now is for the partially optimized PBGS; A’B and A’B’ to obtain A’ PBG. A’ and AC PBGs will be processed to obtain C PBG. The Final result is (A’+C). Note that in the final step A’ PBG is in inclusion relation with AC PBG therefore the eliminated literal will be A of AC PBG to obtain C with keeping of PBG of smallest number of edges, i.e., A’; consider inclusion function in Algorithm 2.

Table 4. Truth table example

Seq.	A	B	C	O
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Table 5. K-Map

		BC		
A	00	01	11	10
0	1	1	1	1
1	0	1	1	0

2.5. LCOA and K-Map of four literal

Consider the truth table presented in Table 7 and its SOP,O=ab'c'd+ab'cd'+ab'cd+abc'd'+abc'd+abcd' +abcd. Also, consider Table 8 and Table 9 which presented the optimization depending on K-map and LCOA respectively. Table 9 includes abstracted bipartite graphs of PBGs to show the ability of LCOA to visualize the solution steps.

According to Table 9, LCOA initially fetches two PBGs according to the inclusion property; ABCD and ABCD', then the proposed ANDing will be applied to obtain ABC PBG. The ABCD and ABCD' PBGs will be neglected. Then LCOA will elite ABC’D' and ABC'D PBGs to obtain ABC'. Then it manipulates AB’CD’ and AB’CD PBGs to obtain AB’C PBG and it ignores AB’CD’ and AB’CD PBGs. The turn now is for the partially optimized PBG ABC' and AB’C'D to obtain AC’D PBG with the elimination of AB’C'D. In the next iteration, LCOA will manipulate AB and AB'C PBGs to obtain AC PBG with ignoring the AB'C

PBG. The Final iteration will manipulate AC and AC'D to obtain AD PBG and neglect the AC'D PBG. The optimized result will be O=AB+AC+AD.

Table 6. LCOA example

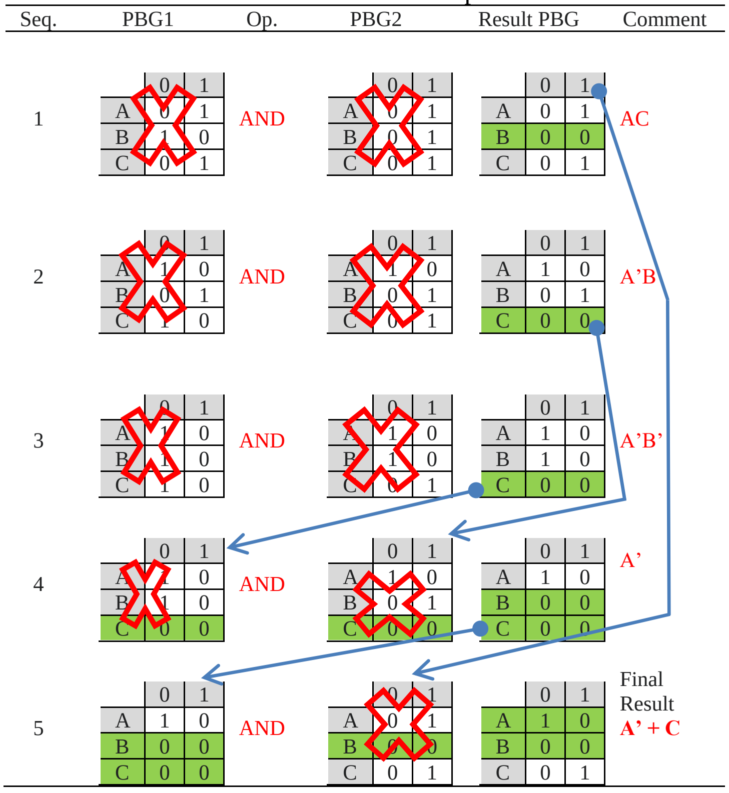


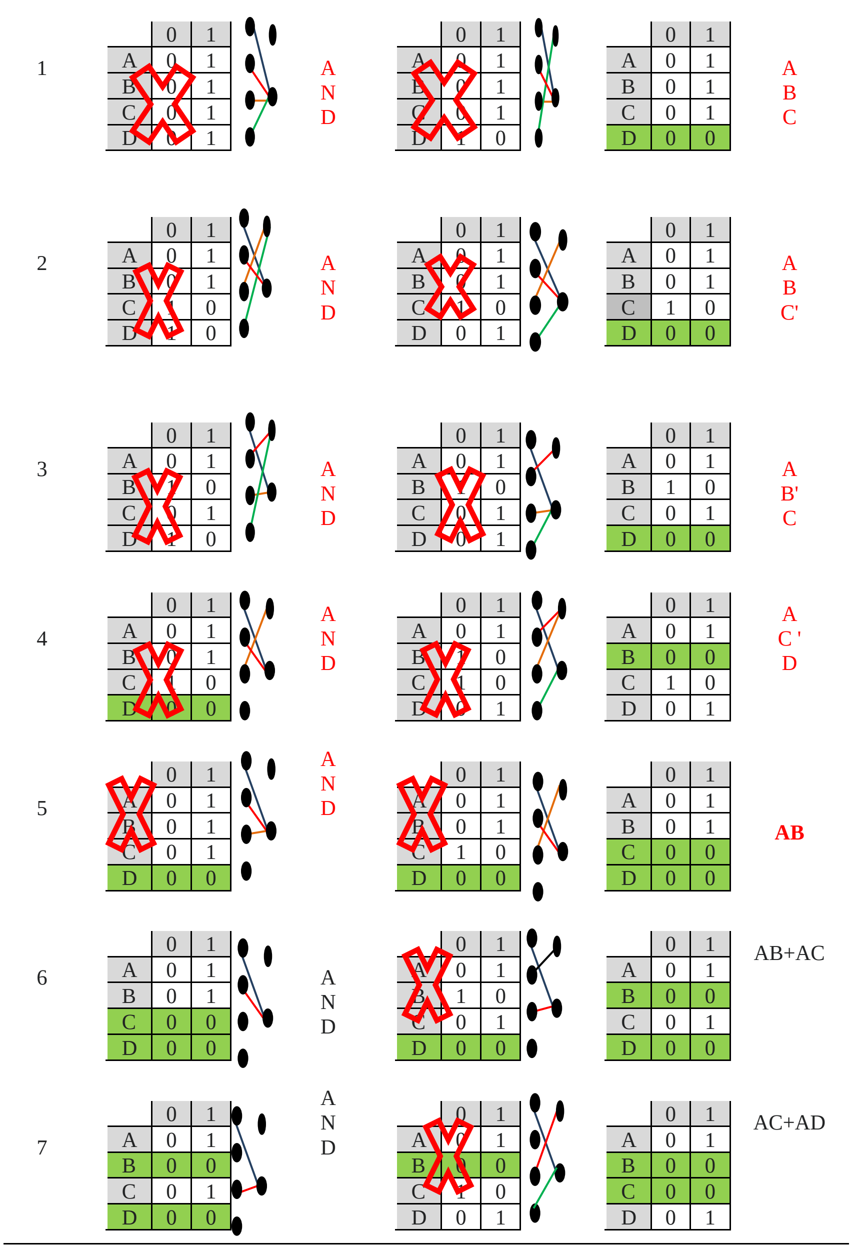
Table 7. Truth table example

Seq.	A	B	C	D	O
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

Table 8. K-Map

CD AB	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	1	1	1

Table 9. LCOA 4 literals example



Seq.	PBG1	G1	Op.	PBG2	G2	Result PBG	Comment
1	0 1 A 0 1 B 0 1 C 0 1 D 0 1		AND	0 1 A 0 1 B 0 1 C 0 1 D 1 0		0 1 A 0 1 B 0 1 C 0 1 D 0 0	A B C
2	0 1 A 0 1 B 0 1 C 1 0 D 1 0		AND	0 1 A 0 1 B 0 1 C 1 0 D 0 1		0 1 A 0 1 B 0 1 C 1 0 D 0 0	A B C'
3	0 1 A 0 1 B 1 0 C 0 1 D 1 0		AND	0 1 A 0 1 B 1 0 C 0 1 D 0 1		0 1 A 0 1 B 1 0 C 0 1 D 0 0	A B' C
4	0 1 A 0 1 B 0 1 C 1 0 D 0 0		AND	0 1 A 0 1 B 1 0 C 1 0 D 0 1		0 1 A 0 1 B 0 0 C 1 0 D 0 1	A C ' D
5	0 1 A 0 1 B 0 1 C 0 1 D 0 0		AND	0 1 A 0 1 B 0 1 C 1 0 D 0 0		0 1 A 0 1 B 0 1 C 0 0 D 0 0	**AB**
6	0 1 A 0 1 B 0 1 C 0 0 D 0 0		AND	0 1 A 0 1 B 1 0 C 0 1 D 0 0		0 1 A 0 1 B 0 0 C 0 1 D 0 0	AB+AC
7	0 1 A 0 1 B 0 0 C 0 1 D 0 0		AND	0 1 A 0 1 B 0 0 C 1 0 D 0 1		0 1 A 0 1 B 0 0 C 0 0 D 0 1	AC+AD

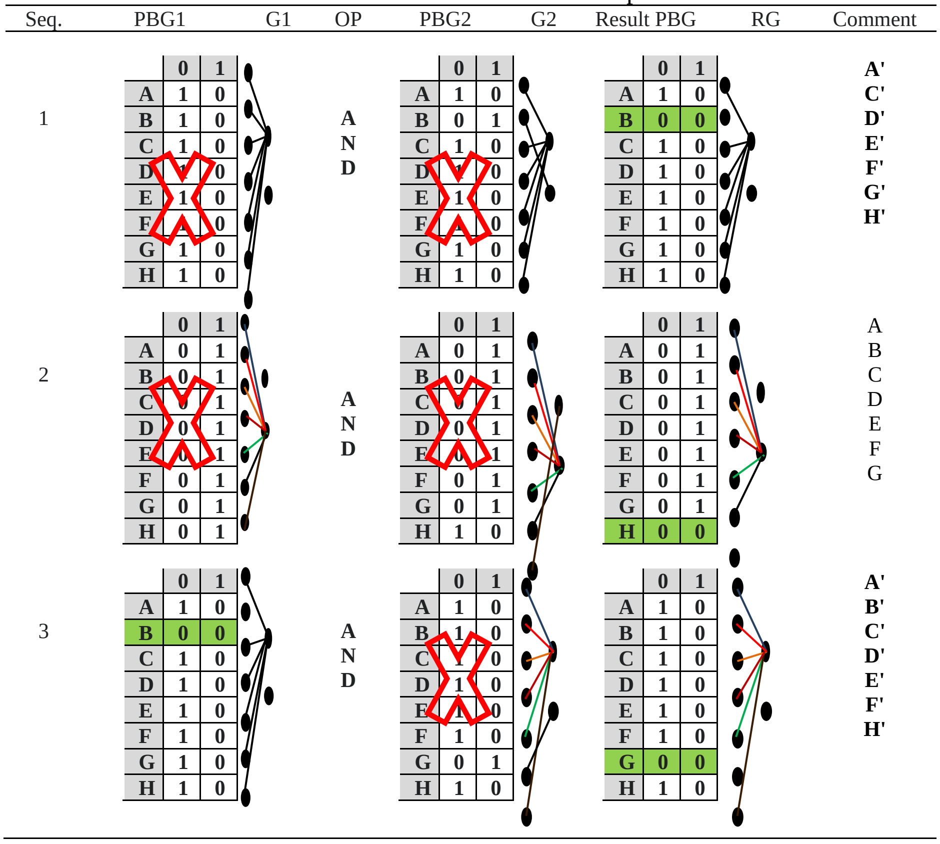
2.6. LCOA and a function of eight literals

Consider the following SOP of eight literals. Also, consider Table 10 which includes the steps of LCOA and abstracted graphs of PBGs. Presenting the abstracted bipartite graph aims to provide well understandability for LCOA, and to illustrate the ability to visualize its steps. Each bipartite graph consists of the set of literals {A, B, C, D, E, F, G and H}, and the set of binary values {0, 1}. The comment field in Table 10 Contains the optimized PBG in each step.

2.7. LCOA's manipulation of don't care and logic 0 output

In LCOA, the don't-care cases can easily be utilized to optimize the logic circuits, this can be accomplished by separating the list of truth PBGs from the list of don't care PBGs. A PBG from the first list will be not checked with the PGGs of the don't-care list except in the situation that no inclusion for it in the truth list. When the truth list becomes in the optimized situation, the don't-care list will be ignored. Let's explain this issue in the following example. Consider the truth table of a segment of the 7-segments display which is presented in Table 11.

Table 10. LCOA 8 literals example



Seq.	PBG1	G1	OP	PBG2	G2	Result PBG	RG	Comment
1	A 1 0, B 1 0, C 1 0, D 1 0, E 1 0, F 1 0, G 1 0, H 1 0		AND	A 1 0, B 0 1, C 1 0, D 1 0, E 1 0, F 1 0, G 1 0, H 1 0		A 1 0, B 0 0, C 1 0, D 1 0, E 1 0, F 1 0, G 1 0, H 1 0		A' C' D' E' F' G' H'
2	A 0 1, B 0 1, C 0 1, D 0 1, E 0 1, F 0 1, G 0 1, H 0 1		AND	A 0 1, B 0 1, C 0 1, D 0 1, E 0 1, F 0 1, G 0 1, H 1 0		A 0 1, B 0 1, C 0 1, D 0 1, E 0 1, F 0 1, G 0 1, H 0 0		A B C D E F G
3	A 1 0, B 0 0, C 1 0, D 1 0, E 1 0, F 1 0, G 1 0, H 1 0		AND	A 1 0, B 1 0, C 1 0, D 1 0, E 1 0, F 1 0, G 0 1, H 1 0		A 1 0, B 1 0, C 1 0, D 1 0, E 1 0, F 1 0, G 0 0, H 1 0		A' B' C' D' E' F' H'

O=a'b'c'd'e'f'g'h'+a'bc'd'e'f'g'h'+abcdefgh+abcdefgh'+.a'b'c'd'e'f'gh'.
The optimized expression is O=a'c'd'e'f'g'h'+abcdefg+a'b'c'd'e'f'h'.

Table 11. A segment of BCD to SSD

Seq.	BCD inputs					Boolean Logic 1	Boolean Logic 0
Decimal	A	B	C	D	a		
0	0	0	0	0	1	A'B'C'D'	
1	0	0	0	1	0		A'B'C'D
2	0	0	1	0	1	A'BC'D'	
3	0	0	1	1	1	ABC'D'	
4	0	1	0	0	0		A' BC'D'
5	0	1	0	1	1	AB'C D'	
6	0	1	1	0	1	A'BCD'	
7	0	1	1	1	1	ABCD'	
8	1	0	0	0	1	A'B'C'D	
9	1	0	0	1	1	AB'C'D	
10	x	x	x	x	0	AB'CD'	Don't Care Cases
11	x	x	x	x	0	AB'CD	
12	x	x	x	x	0	A B C' D'	
13	x	x	x	x	0	ABC'D	
14	x	x	x	x	0	ABCD'	
15	x	x	x	x	0	ABCD	

The full Boolean expression for segment ‘a’ of the display that obtained from Table 11 is:

a=A'B'C'D'+A'BC'D'+ABC'D'+AB'CD'+A'BCD'+ABCD'+A'B'C'D+AB'C'D.

LCOA without caring to don't-care cases produces the following Boolean expression:

a=BD'+B'C'D+A'B'C'+ACD'.

This expression has been obtained in the same manner of the previous examples.

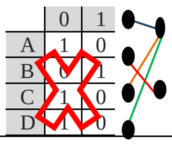
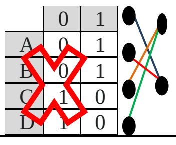
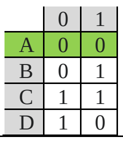
For simplicity, let's use another strategy provided by LCOA that is constructing PBGs that cause a logic 0 output instead of a logic 1 output. However, this strategy gets logic 1 output where a should be at logic zero for A'B'C'D and A'BC'D', therefore this output should be negated to provide correct output that matches the truth table. To manipulate and utilize the don't-care cases, LCOA constructs two lists; list L which contains the PBGs that provide logic 1 or logic 0 and the list of don't care PBGs, DCL. So, L={A'B'C'D, A'BC'D'} and DCL={AB'CD', AB'CD, ABC'D', ABC'D, ABCD', ABCD}

The output of LCOA without considering the don't-care case will be: a=A'B'C'D+A'BC'D' because there is no "*equality*" or "*partial*" inclusion between the PBGs of L. When it is required to LCOA to consider the don't care case, it will fetch A'B'C'D and A'BC'D' but there is no inclusion property, then A'B'C'D will be checked with DCL list for inclusion but there is no inclusion property too, therefore, A'B'C'D will be involved in the output. A'BC'D' PBG is in an equality inclusion with ABC'D' in the DCL and after Anding operation the resultant PBG is BC'D', i.e. a=A'B'C'D+BC'D' which is equivalent to K-map presented in Table 12. The LCOA minimization process is presented in Table 13.

Table 12. K-Map of logic 0

CD AB	00	01	11	10
00	1	0	1	1
01	0	1	x	x
11	**x**	**x**	**x**	**x**
10	1	**1**	**x**	**1**

Table 13. K-Map of logic

Seq.	PBG1 L	Op.	PBG2 DC-List	Result PBG	Comment
1		A N D			B C' D'

3. CONCLUSION

In this paper, a novel algorithm for logic circuit optimization has been produced based on bipartite graph with proposed operations related to bipartite graph such as finding the intersection, union, and/or difference of bipartite graphs. This algorithm excludes the truth table narrative and K-map limitations. Also, it lends itself to programming and easiness of visualization and understanding because it depends on the repetition of excluding the different edges among the bipartite graphs that represent the terms of SOP. Additionally, its output minimization is guaranteed to be global minimum. LCOA can minimize logic circuits using logic 0 output and logic 1 output in the same procedure and equivalent efficiency. This algorithm has been implemented as a program its input is a truth table, terms of SOP, and/or minterms numbers with one or more output.

REFERENCES

[1] D. S. Marakkalage, E. Testa, H. Riener, A. Mishchenko, M. Soeken, and G. De Micheli, “Three-Input gates for logic synthesis,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 10, pp. 2184–2188, Oct. 2021, doi: 10.1109/TCAD.2020.3032625.
[2] K. Wick, *Applied Digital Logic Exercises Using FPGAS*, IOP Publishing, 2017.
[3] S. Bhattacharya, “Truth table analysis of logic circuits using reversible gates,” *International Journal for Research in Applied Science and Engineering Technology*, vol. 8, no. 2, pp. 547–555, Feb. 2020, doi: 10.22214/ijraset.2020.2085.
[4] L. E. Frenzel Jr, *Practical electronic design for experimenters*, McGraw-Hill Education, 2020.
[5] N. Johnson-Glauch, D. S. Choi, and G. Herman, “How engineering students use domain knowledge when problem-solving using different visual representations,” *Journal of Engineering Education*, vol. 109, no. 3, pp. 443–469, Jul. 2020, doi: 10.1002/jee.20348.

[6] W. V. Quine, “The problem of simplifying truth functions,” *The American Mathematical Monthly*, vol. 59, no. 8, pp. 521-531, Oct. 1952, doi: 10.2307/2308219.
[7] C. W. Kann, *Digital circuit projects: an overview of digital circuits through implementing integrated circuits - second edition*, Gettysburg College Open Educational Resources, 2014.
[8] W. V. Quine, “A way to simplify truth functions,” *The American Mathematical Monthly*, vol. 62, no. 9, pp. 627–631, Nov. 1955, doi: 10.1080/00029890.1955.11988710.
[9] H.-G. Vu, N.-D. Bui, A.-T. Nguyen, and ThanhBangLe, “Performance evaluation of Quine-McCluskey method on multi-core CPU,” in *2021 8th NAFOSTED Conference on Information and Computer Science (NICS)*, Dec. 2021, pp. 60–64, doi: 10.1109/NICS54270.2021.9701506.
[10] E. D. Nugroho, “Development of applications for simplification of boolean functions using Quine-McCluskey method,” *Telematika*, vol. 18, no. 1, pp. 27-36, Mar. 2021, doi: 10.31315/telematika.v18i1.3195.
[11] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. *Sangiovanni-Vincentelli*, *Logic minimization algorithms for VLSI synthesis*, vol. 2, pp. 1-194, NY: Springer, 1984, 10.1007/978-1-4613-2821-6.
[12] M. Bolton, *Digital systems design with programmable logic*, Addison-Wesley Longman Publishing Co., Inc, 1990.
[13] R. L. Rudell, “Multiple-valued logic minimization for PLA synthesis,” *Memorandum No. UCB/ERL M86-65*, no. June, 1986.
[14] A. Frieze and M. Karoński, *Introduction to Random Graphs*, Cambridge University Press, 2015.
[15] A. Azaizeh, R. Hasni, F. Haddad, M. Alsmadi, R. Alkhasawneh, and A. Hamad, “4-total edge product cordial for some star related graphs,” *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 12, no. 4, pp. 4007-4020, Aug. 2022, doi: 10.11591/ijece.v12i4.pp4007-4020.
[16] A. S. Asratian, T. M. J. Denley, and R. Häggkvist, *Bipartite Graphs and their Applications*, Cambridge University Press, 1998.
[17] J. van den Brand *et al.*, “Bipartite matching in nearly-linear time on moderately dense graphs,” in *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, Nov. 2020, pp. 919–930, doi: 10.1109/FOCS46700.2020.00090.
[18] A. A. Naem and N. I. Ghali, “Optimizing community detection in social networks using antlion and K-median,” *Bulletin of Electrical Engineering and Informatics (BEEI)*, vol. 8, no. 4, pp. 1433–1440, Dec. 2019, doi: 10.11591/eei.v8i4.1196.
[19] Z. Wu, C. Song, Y. Chen, and L. Li, “A review of recommendation system research based on bipartite graph,” *2020 2nd International Conference on Computer Science Communication and Network Security (CSCNS2020)*, Feb. 2021, vol. 336, doi: 10.1051/matecconf/202133605010.
[20] X. Zhang, H. Wang, J. Yu, C. Chen, X. Wang, and W. Zhang, “Bipartite graph capsule network,” *World Wide Web*, 2022, doi: 10.1007/s11280-022-01009-2.
[21] S. A. Sadchikova and M. Abdujapparova, “Characteristic’s analysis of associative switching system,” *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 20, no. 1, pp. 27-33, Feb. 2022, doi: 10.12928/telkomnika.v20i1.17640.
[22] B. R. Arunkumar and R. Komala, “Applications of bipartite graph in diverse fields including cloud computing,” *International Journal Of Modern Engineering Research (IJMER)*, vol. 5, no. 7, 1-7, 2015.
[23] G. A. Pavlopoulos, P. I. Kontou, A. Pavlopoulou, C. Bouyioukos, E. Markou, and P. G. Bagos, “Bipartite graphs in systems biology and medicine: a survey of methods and applications,” *GigaScience*, vol. 7, no. 4, pp. 1–31, Apr. 2018, doi: 10.1093/gigascience/giy014.
[24] X. Zhang, M. Nadeem, S. Ahmad, and M. K. Siddiqui, “On applications of bipartite graph associated with algebraic structures,” *Open Mathematics*, vol. 18, no. 1, pp. 57–66, Mar. 2020, doi: 10.1515/math-2020-0003.
[25] L. C. C. Bergamasco, K. R. P. S. Lima, C. E. Rochitte, and F. L. S. Nunes, “A bipartite graph approach to retrieve similar 3D models with different resolution and types of cardiomyopathies,” *Expert Systems with Applications*, vol. 193, p. 116422, May 2022, doi: 10.1016/j.eswa.2021.116422.

BIOGRAPHIES OF AUTHORS

Oday Ahmed Al-Ghanimi received the B.Eng. degree in software engineering from Al-Rafidain University College, Baghdad, Iraq, in 2004. He is currently a master's student in computer science at the Institute of Informatics for Graduate Studies, Baghdad, Iraq. He can be contacted at email: ms202030612@iips.icci.edu.iq.

Hussein K. Khafaji received the B.Sc., M.Sc., and Ph.D. degrees from the University of Technology, Baghdad, Iraq, in 1989, 1992, and 2002, respectively. He has been a professor of AI and data mining at Al-Rafidain University College since 2012. He is currently the Head of the Computer Communications Engineering Dept. at Al-Rafidain University College, Baghdad, Iraq. He is also a member of the advisory council of the Iraqi Commission for Computers and Informatics in the Ministry of Higher Education and Scientific Research in Iraq. He has published more than 70 refereed journal and conference papers in the fields of AI and data mining and its applications. He can be reached at hussain.ketan.elc@ruc.edu.iq.