An algorithm for minimizing of Boolean functions based on graph data structure

Masoud Nosrati [*1], Ronak Karimi [2], Hamed Nosrati [3], Ali Nosrati [4]

[1, 2] Young Researchers Club, Kermanshah Branch, Islamic Azad University, Kermanshah, Iran.
[3, 4] Islamic Azad University, Kermanshah Branch, Kermanshah, Iran.
minibigs_m@yahoo.co.uk

Abstract: In this paper, we intend to introduce a new heuristic algorithm to apply maximum minimization to Boolean functions with normal SOP form. To implement the proposed algorithm, we use the graph data structure and define the adjacencies. Also, we demonstrate some conditions to achieve the maximum minimization. Through this paper, the problem of shared vertices in more than one adjacency is talked, and the solution is presented. Karnaugh map is used to clarify the matter.

[Masoud Nosrati, Ronak Karimi, Hamed Nosrati, Ali Nosrati. An algorithm for minimizing of Boolean functions based on graph data structure. Journal of American Science 2011;7(6):92-96]. (ISSN: 1545-1003). http://www.americanscience.org.

Keywords: Minimization of Boolean functions; Graph data structure; SOP functions; discrete mathematics.

1. Introduction

Minimization of Boolean functions is one of the basic operations in Boolean algebra [1]. This is also useful in digital circuits design [2], and it was been regarded to decrease the price of manufactured circuits by removing extra gates [3,4]. In this paper, we present an algorithm to minimize the Boolean functions extremely. We use the graph data structure to implement this algorithm.

Before it, some methods and algorithms was introduced like "Factoring Boolean functions using graph partitioning" [5] or "A Heuristic Method of Two-Level Logic Synthesis" [6]. These methods are absolutely heuristic, and they don't give the maximum minimized form of Boolean function all the time. The method that we are going to introduce is a simple way to reach the maximum minimized form of Boolean function. Also, it could be used in education, because of its simplicity.

In second part which is entitled as "Graph data structure and agreements", the structure of proposed graph and its objects and methods will be talked. In addition, some agreements are presented which are considered during this paper. In third part that is named "SOP functions and graph", the relationship between SOP functions and graph data structure is objected. Furthermore, the conditions of minimization of function by proposed graph are demonstrated. In "Minimization algorithm", that is forth part of this paper, the algorithm of minimization and its description is presented. Eventually, "conclusion" is places as fifth part.

2. Graph data structure and agreements

First time, graph has been used for solving the classic problem of Königsberg bridges by Leonhard Euler in 1736. After that, graph came into mathematics world [7].

A graph is constructed of two sets, *V (vertices)* and *E (Edges)*. For example, look at Figure 1.

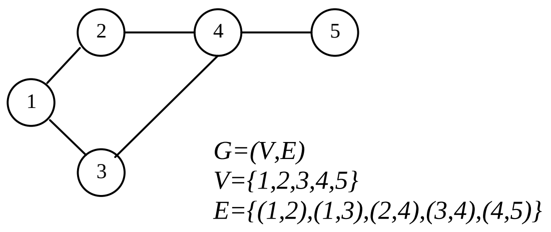

Figure 1. A simple example of graph

A path in graph is a set of vertices we should cross to get to a special vertex. If the initial and final vertices are the same, this path is called cycle, and if all the edges in a cycle are met just one time, it is called a simple cycle [8,9,10].

According to these definitions, there is one simple cycle in Figure 1, which is *{1,2,4,3}*. Here, we make two agreements and describe the reason in third part of paper.

Agreement1: Each vertex makes a simple cycle by itself.

Agreement2: A couple of adjacent vertices make a simple cycle.
(Adjacent vertices are those which are related by an edge.)

According to these agreements, the simple cycles for Figure 1 are like below:

{1} , {2} , {3} , {4} , {5}
{1,2} , {1,3} , {2,4} , {3,4} , {4,5}
{1,2,4,3}

Now, we can implement the class of proposed graph data structure [11,12]. This class contains some objects to store *V* and *E*, and also some methods to create and remove vertices and edges [13,14]. In addition, there is a method that returns the list of all simple cycles which begins with V_i. Another method is defined to return the number of all adjacent vertices of V_i, too.

```
Class Graph
{
   //Objects
         // Data containers to store Vertices and Edges

   public:
         Graph();
               // To create an empty graph
         bool IsEmpty();
               // If graph has no vertices returns TRUE(1), else
               returns FALSE(0)
         void AddVertex(Vertex V);
               // Insert a new vertex
         void AddEdge(Vertex U , Vertex V);
               // Insert a new edge between u and v
         void RemoveVertex (Vertex V);
               // Deletes v and all edges incident to it
         void RemoveEdge(Vertex U , Vertex V);
               // Deletes edge (u,v)
         list Cycles(Vertex Vi);
               // Returns the list of all cycles that begins with  Vi
         int AdjacentVertices(Vertex Vi);
               // Returns the number of adjacent vertices of Vi
}
```

3. SOP functions and graph

Boolean functions are used for indicating the performance of complex two-level circuits with AND - OR gates. These functions could be shown in normal SOP (Sum Of Products) or POS (Product of Sums) forms [15,16,17]. We aren't going to talk about algebraic concepts or the way of generating SOP or POS forms. Just propose that we have a SOP function which should be minimized.

There are different ways to minimize SOP functions. One is using the algebraic rules, which is hard and confusing for large functions with many variables. Another is using Karnaugh map. It could be used for functions with 2 to 6 variables, by drawing the map of adjacency. In fact, Karnaugh map is an illustrative form of truth table. It puts the adjacent statements near each other and provides the opportunity of selecting appropriate adjacency.

Figure 2 shows the Karnaugh map for 4-variables Boolean functions. In this map, different states of variables are showed by 0 and 1 [18].

0000	0001	0011	0010
0100	0101	0111	0110
1100	1101	1111	1110
1000	1001	1011	1010

Figure 2. Karnaugh map for 4-variables Boolean functions

It is seen that each two adjacent cells has one different bit. In other word, the XOR of two adjacent cells equals 2^r, $r=0,1,2,\ldots$.

Consider that function (I) should be minimized by this map. By replacing the variables with 0 and 1 (function (II)), its Karnaugh map will be as Figure 3.

(I)
$f(w,x,y,z) = w'x'yz' + w'xy'z + w'xyz' + w'xyz + wx'yz' + wx'yz + wxyz'$

(II)
$f(w,x,y,z) = 0010 + 0101 + 0110 + 0111 + 1010 + 1011 + 1110$

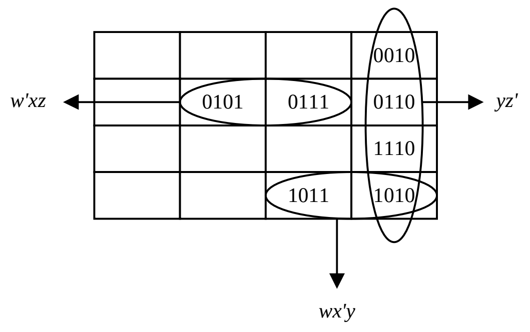

Figure 3. Karnaugh map for function (I) with appropriate adjacencies

In Figure 3, appropriate adjacencies are selected, and minimization operation - which is remaining similar bits and removing the others [19] - is done. Essential condition to choose an appropriate adjacency is defined as (*).

(*)

The number of cells in an adjacency should be equal to 2^k, *k=0,1,2,…* and no similar bits equal to *k*.

Another point which should be regarded is that always the biggest adjacencies that contain more cells should be selected, in order to make the function more minimized, by reducing more different bits [1,18]. Also, it should be paid attention that in functions with complete statements - where all statements are present - minimized function equals to 1.

It is seen that minimized function (I) will be like function (III).

(III)
$f(w,x,y,z) = yz' + w'xz + wx'y$

Suppose that f_v is a desire Boolean function. It could be adapted to graph data structure, if for each statement in it, create a vertex and show adjacencies by edges. For example, Figure 4 shows the graph of function (I).

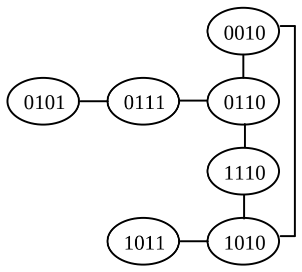

Figure 4. Graph of function (I)

To minimize the function of this graph, first the biggest appropriate adjacencies should be found for each vertex. It has to be done regarding agreements 1 & 2. Now, you can find out the reason of making these agreements. Minimization is operated according to the adjacencies (not vertices), so for each alone vertex, an adjacency should be considered. For two adjacent vertices it has to be done, too. In mathematical definition of simple cycles in no directed graphs, simple cycles with less than 3 vertices are not defined [5,9].

Regarding condition (*), Table 1 shows the biggest simple cycles in graph of Figure 4.

Vertex	Simple cycle *(*)*	Minimized
0010	0010-0110-1110-1010-(0010)	*yz'*
0101	0101-0111-(0101)	*w'xz*

0111	0111-0101-(0111)
	0111-0110-(0111)
		w'xz
		w'xy

0110	0110-1110-1010-0010-(0110)	*yz'*
1110	1110-1010-0010-0110-(1110)	*yz'*
1011	1011-1010-(1011)	*wx'y*
1010	1010-0010-0110-1110-(1010)	*yz'*

Table 1. List of biggest cycles of graph Figure 4 regarding condition (*) for each vertex

For vertices *0010, 0110, 1110, 1010* cycles are the same. Consequently, the minimized forms are similar, too. For vertex *0111* two appropriate adjacency is available. In other word, the biggest cycle is not unique. If both of them be involved in final minimized function, then one extra statement is imposed to it. So, one of them must be chosen as below:

If vertex *V* has more than one biggest cycle regarding (*), choose the adjacency that its first vertex (next to proposed *V*) has less adjacent.

In our example, vertex *0111* has two adjacent *0101* and *0110*. First one has 1 adjacent vertex, and second has 3. So, first one has to be chosen and *w'xz* should appear in final minimized rather than *w'xy*.

The reason is when you choose the path which its first vertex has less adjacent, probability for this vertex to be included in other adjacencies is less, and if it has no other adjacent vertices, it couldn't participate in minimization operation. So, to make sure that it never happens, choose the path with less. But, if the number of adjacent vertices for both of them was equal, then you can choose one randomly. Because, they have similar circumstances and it doesn't differ that which one is selected.

4. Minimization algorithm

To implement the maximum minimization on the introduced graph, the algorithm in below is offered.

```
\\ -------------------------------- Minimization algorithm based on graph data structure -----------------------------

Create Graph of Boolean function;
If all the vertices and edges are present then return 1;
Else
{
        For each vertex V
        {
                Find biggest cycles with the condition (*);
                If biggest cycle is not unique then
                {
                        Find the num of adjacent vertices of first vertex next to V in path;
                        If the numbers of adjacent vertices are equal then
                                Choose one randomly;
                        Else
                                Select the path with lest adjacent vertices for its first vertex;
                }
                Minimize (take the similar bits and reduce others);
                Store the minimized forms;
        }
        Reduce the repeated minimized statements;
        Return the minimized function;
}
```

First, it creates a graph according to Boolean function. Then, it checks whether the function is complete, return 1. Else, find the biggest cycles for each vertex, and if it wasn't unique, choose the appropriate one. After that, minimizes the adjacency by taking the similar bits and reduce others. Then, it stores the minimized form. When these steps were done for all the vertices, some statements will be created per vertices (as you see in Table 1). After reducing the repeated ones, final minimized function will be achieved and returned.

5. Conclusion

In this paper, we introduced a new heuristic algorithm to apply maximum minimization to SOP Boolean functions. Therefore, graph data structure as the essential base of this algorithm was defined, and two agreements were made which forked from the difference between mathematical definition of simple cycles and what we need for our object. Then, the method of minimization was talked, which used the concepts of Karnaugh map. Finally, the algorithm of minimization presented.

Corresponding Author
Masoud Nosrati
Department of Computer Engineering
Islamic Azad University, Kermanshah Branch, Young Researchers Club, Kermanshah, Iran.
E-mail: minibigs_m@yahoo.co.uk

References

1. T. Sasao. EXMIN2: A Simplication Algorithm for Exclusive-OR-Sum-of -Products Expression for Multiple-Valued-Input Two-Valued-Output functions. IEEE Trans. on Computer Aided Design 12 (1993) 621-632.
2. David Money Harris, Sarah L. Harris, Digital Design and Computer Architecture, Morgan Kaufmann, 2007, pp. 51-62.
3. M.A. Thornton, R. Drechsler, and D.M. Miller. Spectral Techniques in VLSI CAD. Kluwer Academic Publ., 2001.
4. Morris M. Mano, Digital Design, 4th ed, Prentice Hall, 2006, pp. 36-110.
5. Aviad Mintz, Martin Charles Golumbic, Factoring Boolean functions using graph partitioning, Discrete Applied Mathematics 149 (2005) 131 – 153
6. Jan Hlavička, Petr Fišer, A Heuristic Method of Two-Level Logic Synthesis, Karlovo nám. 13, 121 35 Prague 2.
7. Elis Horowitz, Sartag Sahni, Dinish Mehta, Fundamentals of Data Structures in C++, 2nd ed, Silicon Press, 2006.
8. John Adrian Bondy, U. S. R. Murty, Graph theory with applications, 9th ed, Elsevier Science Ltd, 1976, pp. 1-24.
9. Seymour Lipschutz, Schaum's outline of theory and problems of discrete mathematics, 3rd ed, McGraw-Hill, 2009, pp. 154-200.

10. M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York, 1980. Annals of Discrete Mathematics, second ed., vol. 57, Elsevier, Amsterdam, 2004.
11. Sedgewick, Robert, Algorithms, Consulting Editor Harrison, Michael A., Addison-Wesley, 1984. pp. 373-455.
12. Seymour Lipschutz, Schaum's outline of theory and problems of data structures, Mcgraw-Hill, 1986.
13. Ian Parberry, Lecture notes on algorithm analysis and computational complexity (ebook), Department of Computer Science, University of North Texas. Pages 66 to 71.
14. Martin Charles Golumbic, Algorithm graph theory and perfect graphs, second edition, Elsevier, 2004, pp. 31-37.
15. Balch, Mark, Complete Digital Design, McGraw-Hill, 2003. pp. 3-32.
16. Popel, Denvis V., Information theoretic approach to logic function minimization (ebook), Technical University of Szczecin, 2000.
17. Yuke Wang, Data structures, Minimization and complexity of boolean functions (ebook), A thesis of Ph.D degree, University of Saskatchewan, Canada (1995). Pages 8 to 20.
18. Victor Peter Nelson, Digital logic circuit analysis and design, 2nd sub ed, Prentice Hall, 1995, pp. 90-120.
19. R. M. Karp Reducibility Among Combinatorial Problems. in R. E. Miller and J. W. Thatcher (editors): Complexity of Computer Computations. New York: Plenum Press (1972), 85103.

5/7/2011