MINIMIZATION OF BOOLEAN FUNCTIONS USING GENETIC ALGORITHM

Masoud Nosrati, Ronak Karimi, Mehdi Hariri

Kermanshah University of Medical Sciences, Kermanshah, Iran

ABSTRACT: Minimization of Boolean functions is one of basic Boolean algebra functions. This paper presents a method for minimizing Boolean functions. To do this, first a graph data structure that is needed for storing Boolean function and basic operations will be investigated.In fact, it is used for storing Karnaugh map adjacencies. Then, the adjacencies and conditions for selection of appropriate adjacencies for factoring are nominated. As a essential part of paper, a brief review of genetic algorithms is presented and finally usage of GA for selection of appropriate adjacencies is described.
KEYWORDS: Boolean Functions, SOP, Minimization, Factoring, Genetic Algorithm, GA.

1. INTRODUCTION

Minimization of Boolean functions is one of the basic operations in Boolean algebra [NKA11]. This is also a practical step in two-level digital circuits design [MMM06, TDM01], and it was been regarded to decrease the price of manufactured circuits by removing extra gates [HH07] by factoring operation. There are some standard techniques used to compute a minimal cost sum-of-products representation of a given Boolean function [Cou94], such as: Karnaugh map method, Quine–McCluskey procedure [B+84], formalizing covering matrix reduction [CMF93], the signature cube based procedure [B+93] and minimization of Boolean functions using GDS [NH11].
In this paper, we are going to introduce a new genetic algorithm based method for presenting the minimal show of a Boolean function. For achieving this aim, storing and specifying the adjacent statements of Boolean function are the basic needs. Due to this, in the second section we will get into the adjacency graph data structure that is used for simulation of Karnough map. Also, finding the adjacencies that a vertex is token part is investigated. Third section will talk about finding appropriate adjacencies for factoring. We will use the genetic algorithm to do it. Fitness function for GA is calculated so that achieve to least number of output Boolean statements. Finally, fourth section is dedicated to conclusion that includes the advantages and features of this method.

2. SPECIFYING THE ADJACENCIES USING GDS

Maurice Karnaugh's 1953 refinement of Edward Veitch's 1952 Veitch diagram is a method to simplify Boolean algebra expressions. The Karnaugh map reduces the need for extensive calculations by taking advantage of humans' pattern-recognition capability, also permitting the rapid identification and elimination of potential race conditions.
In a Karnaugh map the Boolean variables are transferred (generally from a truth table) and ordered according to the principles of Gray code in which only one variable changes in between adjacent squares. Once the table is generated and the output possibilities are transcribed, the data is arranged into the largest possible groups containing 2^n cells *(n=0, 1, 2, 3, ...)* and the minterm is generated through the axiom laws of Boolean algebra [Kar53].
In this study, we will have a Karnaugh approach to Boolean functions. So, first we need an appropriate data structure for storing and operating the Karnaugh map. Nosrati et al. introduce a Graph DS for this purpose [NH11].

2.1 Adjacency 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 [BM76, Lip09, Gol04].
We can consider a Boolean function in SOP form as a set of Boolean statements. Each statement can be a vertex of Graph and each edge can be the sign of adjacent statements (as what there is in Karnaugh map). So, each simple cycle indicates an adjacency for factoring. Here, two agreements are made as follow:

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.)

Proposed Graph data structure class code is listed in follow. This class contains some objects to store *V* and *E*, and also some methods to create and remove vertices and edges [HSM06, Sed84, Lip86, Par71, Gol04]. 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 [NH11].

```
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
}
```

2.2 Mapping the Boolean function to Graph DS

As it was mentioned, 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 1 shows the Karnaugh map for 4-variables Boolean functions. In this map, different states of variables are showed by 0 and 1 [Nel95].

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

Figure 1. Karnaugh map for 4-variables Boolean functions

A Boolean function may include some of the cells or whole of them. The cells that are included by Boolean function are ON (1) cells and the rest are OFF (0). So, in mapping the function to Graph, only the ON cells must be considered.
For creating the set of edges (or adjacencies) it should be regarded that adjacent vertices are those which are different just in a bit. In other words, the *XOR* of two adjacent cells equals 2^r, *r=0, 1, 2, ….*

2.3 Adjacencies specification

As it is said, an adjacency is a set of vertices that can be minimized together in Karnaugh map. There might be some adjacencies in each Boolean function. A condition for selecting an adjacency is as *(*)*. Minimization operation - which is remaining similar bits and removing the others [Kar72] – can be done only on adjacencies that are in accepted by *(*)*. 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* .

In each Karnaugh map, a vertex might be included in some different adjacencies. Appropriate selection of adjacencies is very important. We will use GA for selecting them. But, first, we need a list of all adjacencies. As it is shown in Graph class, there is a method that returns the list of simple cycles for each vertex. If we execute this method for all vertices, and collect the results, we will achieve the list of all adjacencies. Also, repeated adjacencies must be considered one time. Note that all the adjacencies cannot take part in minimization process. Only the adjacencies that are accepted by (*) must be considered and the rest must be rejected. Acceptable adjacencies can be stored in other objects and they can be titled by numbers, alphabets, etc., like *1, 2, 3, ….*

3. GENETIC ALGORITHM FOR SELECTING APPROPRIATE ADJACENCIES

In this section, first we will get into the basic principals of genetic algorithms such as the concepts of population, generation, reproduction, crossover, mutation, etc. Then, in the next part of current section, we use GA for selection of appropriate adjacencies to minimize the Boolean functions.

3.1 Principals of GA

Genetic Algorithms are a family of computational models inspired by evolution. These algorithms encode a potential solution to a specific problem on a simple chromosome-like data structure and apply recombination operators to these structures as to preserve critical information. An implementation of genetic algorithm begins with a population of (typically random) chromosomes. One then evaluates these structures and allocated reproductive opportunities in such a way that these chromosomes which represent a better solution to the target problem are given more chances to 'reproduce' than those

chromosomes which are poorer solutions. The 'goodness' (or fitness) of a solution is typically defined with respect to the current population [Whi94].
The working principle of a canonical GA is illustrated in Figure 2. The major steps involved are the generation of a population of solutions, finding the objective function and fitness function and the application of genetic operators. These aspects are described briefly below. They are described in detail in the following subsection [Mat05].

/*Algorithm GA */
Formulate initial population
Randomly initialize population
Repeat
Evaluate objective function
Find fitness function
Apply genetic operators
Reproduction
Crossover
Mutation
Until *(stopping criteria)*

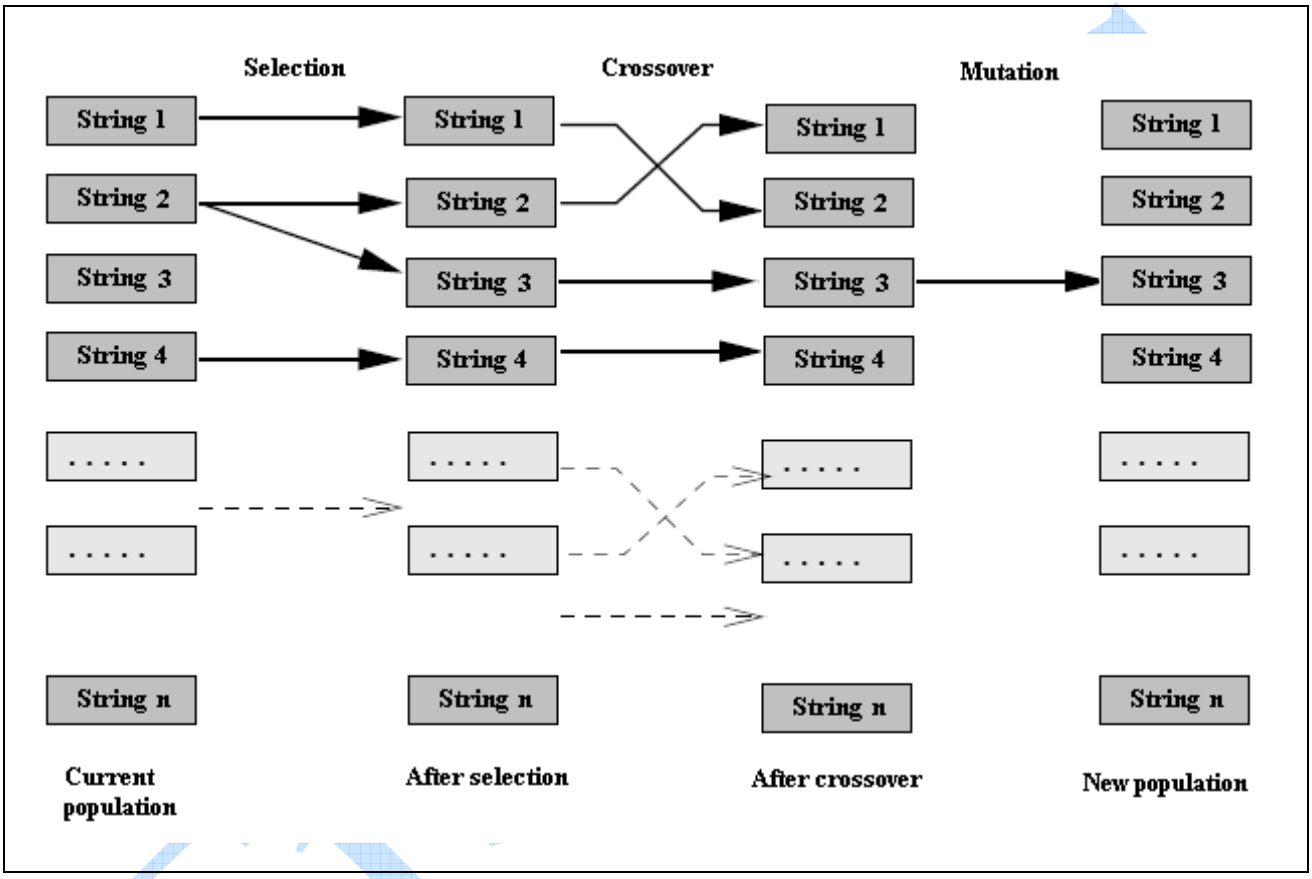

Figure 2. Working principle of a canonical GA from current population to new population

If we choose n chromosomes (strings), after generation of new population we will have $2n$ chromosomes, which include previous and new reproduced chromosomes. Now, n chromosomes must be selected and the others will be removed. Selection of new population is based on fitness function. Fitness function for each chromosome shows its difference to goal chromosome. This algorithm repeats these steps until it gets to a set of repeated chromosomes in a sequence of generations.

3.2 Using GA for selecting appropriate adjacencies

Now, we have a set of adjacencies and we want to choose the best of them for minimization. Initial population can be formed easily. Due to it, we can for example consider n chromosome, so that $n=2^{nw(variables)}$. It means number of chromosomes can be equal to 2 in power of number of variables in Boolean function. Note that amount of n is just an offer and it might be changed.
Most important point about chromosomes is the number of genes in them. You know that in a Karnaugh map, number of adjacencies that are considered is dependent, and it might be from 2^0 to $2^{nw(variables)}$. So, we will have a non-identical population. For example for a Boolean function with 2 variables, we will have a population like Figure 3.
There is an important point in reproduction. Reproduction cannot be done in crossover way. Because of heterogeneous chromosomes, crossover operation cannot operate them. So, just the mutation can be done.

As you know, each gene is the title of an adjacency, and the statements in an adjacency can be minimized together. For calculating the fitness function, we should look at the genes of a chromosome. Fitness can be equal to length of non-repeated Boolean statements that remains after minimizing the genes of a chromosome.

$$Fitness = \sum |MinimizedBooleanStatements|$$

After repeating this algorithm over and over, we will get to a repeated set of chromosomes in a sequence of generations. Now, the result is $2^{nw(variables)}$ best adjacencies. In other words, the number of best found adjacencies are $2^{nw(variables)}$.

Each adjacency may include some of statements and refuse the others. For calculation of total minimized form using supposed adjacency, we must consider whole Boolean function except the statements that are included in supposed adjacency. Instead of them, we add the minimized form of adjacency statements. Fore sure, one of the chromosomes in the last generation contains the best adjacencies. So, after finishing genetic algorithm process, we must calculate the minimized form based on the chromosomes of last generation. After reducing the repeated ones, the best minimized form that is the shortest one will be appeared.

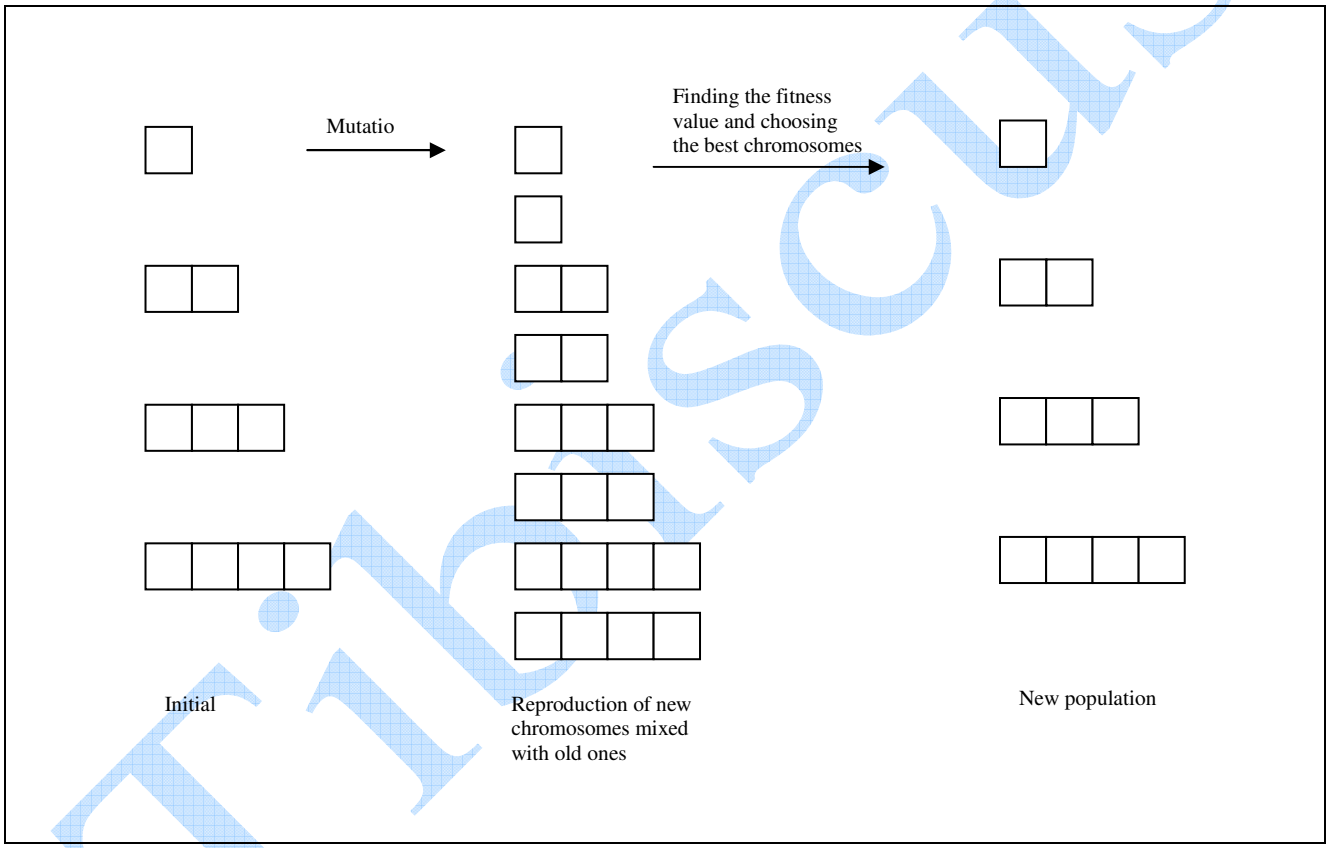

Figure 3. GA with non-identical population

4. CONCLUSIONS

In this paper we got into the problem of minimization of Boolean functions. So, we had a brief look at the concept of adjacency in Karnaugh map and introduced a Graph data structure for storing the Boolean function adjacencies. Then, we extracted the list of all adjacencies and gave it to a genetic algorithm with non-identical chromosomes. The role of GA was finding best adjacencies for minimization. After finishing genetic algorithm, we had $2^{nw(variables)}$ set of adjacencies set. Then the best of them were selected by calculation of minimized form using them. Shortest minimized form showed the best of them and the maximum minimized form of SOP Boolean function.

REFERENCES

[BM76] **J. A. Bondy, U. S. R. Murty** – *Graph theory with applications*, 9th edn. pp. 1–24. Elsevier Science Ltd, Amsterdam, 1976.

[B+84] **R. K. Brayton, G. D. Hachtel, C. T. McMullen, A. L. Sangiovanni-Vincentelli** – *Logic Minimization*

Algorithms for VLSI Synthesis, Kluwer Academic Publishers, Dordrecht, 1984.

[B+93] **R. K. Brayton, P. C. McGeer, J. Sanghavi, A. L. Sangiovanni-Vincentelli** – *A New Exact Minimizer for Two-level Logic Synthesis, in: Logic Synthesis and Optimization*, T. Sasao (Ed.), Kluwer Academic Publishers, Dordrecht, 1993, pp. 1–31.

[Cou94] **O. Coudert** – *Two-Level Logic Minimization: An Overview*, Integration Vol. 17, No. 2, pp. 97-140, Oct. 1994.

[CMF93] **O. Coudert, J. C. Madre, H. Fraisse** – *A New Viewpoint on Two-Level Logic Minimization*, Proc. 30th Design Automation Conference, Dallas, TX, USA, pp. 625–630, June 1993.

[Gol04] **Golumbic, M.C.** – *Algorithmic Graph Theory and Perfect Graphs*, 2nd edn. Annals of Discrete Mathematics, vol. 57. Elsevier, Amsterdam, 2004.

[HH07] **Harris, D.M., Harris, S.L.** – *Digital Design and Computer Architecture*, pp. 51–62. Morgan Kaufmann, San Francisco, 2007.

[HSM06] **Elis Horowitz, Sartag Sahni, Dinish Mehta** – *Fundamentals of Data Structures in C++*, 2nd ed, Silicon Press, 2006.

[Kar53] **Karnaugh, Maurice** – *The Map Method for Synthesis of Combinational Logic Circuits*, Transactions of the American Institute of Electrical Engineers, November 1953, part I 72 (9): 593–599.

[Kar72] **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.

[Lip09] **S. Lipschutz** – *Schaum's outline of theory and problems of discrete mathematics*, 3rd edn. pp. 154–200. McGraw-Hill, New York 2009.

[Lip86] **S. Lipschutz** – *Schaum's outline of theory and problems of data structures*, Mcgraw-Hill, 1986.

[Man06] **M. M. Mano** – *Digital Design*, 4th edn. pp. 36–110. Prentice Hall, Englewood Cliffs, 2006.

[Mat05] **T. V. Mathew** – *Genetic Algorithm*, 2005, Lecture Note, Avalable at: http://www.civil.iitb.ac.in/tvm/2701_dga/2701-ga-notes/gadoc.pdf.

[Nel95] **V. P. Nelson** – *Digital logic circuit analysis and design*, 2nd sub ed, Prentice Hall, 1995, pp. 90-120.

[NH11] **Masoud Nosrati, Mehdi Hariri** – *An Algorithm for Minimizing of Boolean Functions Based on Graph DS*, World Applied Programming, Vol (1), No (3), August 2011. 209-214.

[NKA11] **Masoud Nosrati, Ronak Karimi, Reza Aziztabar** – *Minimization of Boolean Functions Which Include Don't-Care Statements, Using Graph Data Structure*, In Proc. of: WiMoA 2011/ICCSEA 2011, Dubai, UAE. Springer CCIS 154, pp. 212–220, 2011.

[Par71] **Ian Parberry** – *Lecture notes on algorithm analysis and computational complexity*, (ebook), Department of Computer Science, University of North Texas. Pages 66 to 71.

[Sed84] **Robert Sedgewick** – *Algorithms*, Consulting Editor Harrison, Michael A., Addison-Wesley, 1984. pp. 373-455.

[TDM01] **Thornton, M.A., Drechsler, R., Miller, D.M.** – *Spectral Techniques in VLSI CAD*, Kluwer Academic Publ., Dordrecht, 2001

[Whi94] **D. Whitley** – *A genetic algorithm tutorial*, Stat&tics and Computing (1994) 4, 65-85.