Generating Threshold Functions of up to Eight Variables Using Computer

Kasiyah Machmudin
Email: kasiyah@makara.cso.ui.ac.id
The Department of Mathematics, The University of Indonesia

Abstract
Threshold functions are restricted class of Boolean functions that model neurons. In an artificial neural network a neuron represents a processing unit. Threshold functions also attracted some attention because they realize threshold gates, which have higher processing power compared to the conventional gates.

It has been proven that a necessary and sufficient condition for a Boolean function of less than eight variables be a threshold function is being completely monotonic. This property and other theorems shown in this paper are used to develop an algorithm and a computer program that generates threshold functions of up to eight variables.

In order to express threshold functions and their useful properties in a programming language, such as Pascal, a new way of representing threshold functions based on the minterm expansion form via vertex maps is introduced. This representation, which is strongly related to the geometric aspects of threshold functions, seems very fruitful. Some suggestions for future research topics that is related to enhancement of the new generating algorithms and potential uses of vertex maps are presented in this paper. There are three students doing their research in this field.

Although the algorithm developed works for threshold functions with eight variables the program can generate only those of up to five variables because of some limitations. The enumeration of those functions and their non-isomorphic vertex maps are given in this paper.

Key words: threshold function, completely monotonic, minterm expansion form, and vertex map.

1. Introduction

The logical operations of threshold gates can be described in forms of a restricted class of Boolean functions called *threshold functions*. Threshold gates have higher processing power compared to the conventional gates [6]. Moreover, in general a Boolean function can be mechanized with fewer threshold devices than with *AND*, *OR*, and *NOT* gates.

Besides illustrating the logical operation of threshold gates, threshold functions model neurons. Each neuron can be regarded as a processing unit in an artificial neural network. A formal neuron consists of *n* input lines, each of them having a weight, and a single output line. The output lines of the neurons can be connected to some other neurons to forms a neural network.

Definition 1:
A Boolean function f of n variables $X = (x_1, x_2, \ldots, x_n)$ is a threshold function if and only if there exists a set of real numbers $w_1, w_2, \ldots, w_n$, called (*input*) *weights*, and a real number θ, called *the threshold*, such that the following conditions are satisfied:

$$f(X) = 1 \quad \text{if } \sum_{1}^{n} x_i w_i \geq \theta ,$$
$$\qquad\qquad (1)$$
$$f(X) = 0 \quad \text{otherwise.}$$

In that case, [$w_1, w_2, ..., w_n ; \theta$] is called a *structure of f.*

For instance, [1, 1; 2] is a structure of threshold function $f = x_1 x_2$ ($x_1 AND x_2$). In fact, [1, 2; 3] is another structure of *f.* In general, a threshold function can be realized by an infinite number of structures. The logical summation $f = x_1 + x_2$ ($x_1 OR x_2$) is another example of threshold function. However, $f = x_1 \overline{x_2} + \overline{x_1} x_2$ ($x_1 XOR x_2$) is not a threshold function since there is no such a set of real numbers that satisfy the system of inequalities (1). Note that the definition does not provide a procedure how to find a structure of a threshold function. In [13], such structures were determined by solving the system of inequalities by the simplex method.

2. Threshold Function Representations

There are two standard forms that are commonly used to represent a Boolean function: *the minterm expansion form*, abbreviated m.e.f, and *the irredundant disjunctive form*. The representation chosen in this paper is the m.e.f. because it can be adapted in a program easily and it is strongly related to the geometrical representations of functions.

A variable or its complement is called *literal.* A *minterm* is a conjunction of different literals in which each variable is involved exactly once. The minterm expansion form is a disjunction of different minterms. For instance, let $f = x_1 + \overline{x_2}$. The m.e.f. of f is $f = x_1 x_2 + x_1 \overline{x_2} + \overline{x_1}\,\overline{x_2}$. Each Boolean function can be expressed uniquely in the m.e.f.

Minterms appear in the m.e.f. of *f* are called *true* minterms or *minterms of f*, otherwise are called *false* minterms. For simplicity's sake a Boolean function of *n* variables having *m* minterms is called an (*n, m*) Boolean-function.

Given an *n*-vector $X = (x_1, x_2, ..., x_n)$, *A* is a *k*-assignment on *X* if *A* is a mapping from *k*-subset X^A of the variables into {0, 1}. If $k = n$ then *A* is called a complete assignment. Two assignment *A* and *B* can be combined (multiplied) provided that $X^A \cap X^B = \emptyset$; *AB* is the assignment on *X* defined by $AB(x) = A(x)$ if $x \in X^A$ and $AB(x) = B(x)$ if $x \in X^B$. For instance, if $A = \{x_2 = 0, x_3 = 1\}$, and $B = \{x_1 = 1, x_4 = 1\}$, we have $\overline{A} = \{x_2 = 1, x_3 = 0\}$, and $AB = \{x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1\}$. We write f_A to represents the function that is obtained from *f* induced by assignment *A.* If *A* is a complete assignment then f_A can be denoted as *f*(*A*).

Let *f* be a Boolean function, $\mathbf{m} = x_1^* x_2^* ... x_n^*$ be a *true* minterm, and $X_{\mathrm{m}} = (a_1^*, a_2^*, ..., a_n^*)$ be a vector with the following relation: $a_i^* = 1$ if $x_i^* = x_i$, and $a_i^* = 0$ if $x_i^* = \overline{x_i}$. The vector and the minterm are related by the assignment $A_{\mathrm{m}} = \{x_1 = a_1^*, x_2 = a_2^* = ..., x_n = a_n^*\}$. The value of **m** under the assignment A_m is1, hence the value of *f* is 1 also. In other words, the *true* minterms are in one-to-one correspondence with the rows of the truth table for *f* in which $f = 1$; the *false* minterms are in one-to-one relationship with such rows in which $f = 0$. Furthermore, a minterm **m** (or its corresponding vector X_m) is in one-to-one correspondence with the assignment A_m that relates **m** to vector X_m, called a *true* vector. Vectors related to *false* minterms are called the *false* vectors.

Given two complete assignments *A* and *B*. If x_i is mapped to 1 (or 0) in both *A* and *B* then *A* and *B* are *identical in valuation i.* Say that *A* and *B* are identical in *k* out of *n* valuations. These identical valuations can be regarded as a *k*-assignment *S* called the *common assignment* between *A* and *B*. The (*n-k*)-assignments in *A* and in *B* are two complementary assignments called the *differing assignments* between *A* and *B*.

Furthermore, two minterms can be compared in the same way as we compare two complete assignments. If x_i appears complemented (or un-complemented) in both minterms $\mathbf{m}_i$ and $\mathbf{m}_j$ then we say that $\mathbf{m}_i$ and $\mathbf{m}_j$ are *identical* in the *i*-th literal. Suppose that they are identical in *k* out of *n* literals, the product of these literals forms a term called the *identical subminterm* **s** between $\mathbf{m}_i$ and $\mathbf{m}_j$. The (*n-k*) differing literals in $\mathbf{m}_i$ and $\mathbf{m}_j$ form two complementary terms called the *differing subminterms,* **e** and **ē,** between $\mathbf{m}_i$ and $\mathbf{m}_j$. In this case, $\mathbf{m}_i$ and $\mathbf{m}_j$ can be written as **se** and **sē** respectively.

In the program, a minterm $\mathbf{m} = x_1^* x_2^* ... x_n^*$ is represented as a binary number of length *n*, i. e. $b_1^* b_2^* ... b_n^*$, where $b_i^* = 1$ if $x_i^* = x_i$ and $b_i^* = 0$ if $x_i^* = \overline{x_i}$. Thus, a Boolean function *f* can be written as a set of binary numbers. Each binary number in *f* represents the corresponding *true* minterm of *f*. If the binary number is converted into its decimal representation, then a function can also be represented as a set of positive integers. Sets can be manipulated easily in a program. Adding one minterm to *f* means adding a member into the set of *f*. If *f* is given in the m.e.f., the conversion of *f* into the set of *f* is straightforward. For instance, the function $f = \overline{x_1} x_2 x_3 + x_1 \overline{x_2} x_3 + x_1 x_2 x_3$ can be written as follows:

Table 1: The Representations of *f*

The m.e.f. of *f*	The Binary-Number Set of *f*	The Decimal Set of *f*
$f = \overline{x_1} x_2 x_3 + x_1 \overline{x_2} x_3 + x_1 x_2 x_3$	*f* = {011, 101, 111}	*f* = {3, 5, 7}

3. Geometrical Interpretation of Threshold Function

Both *true* and *false* vectors represent corner points (vertices) of an *n*-hypercube. For simplicity's sake, let a vertex be called a vector and vice versa. Thus, an (*n, k*) Boolean-function can be explained in terms of geometry as a unit *n*-hypercube in which each vertex is labeled either by 1 or 0. The *true* vertices (*true* vectors) are labeled by 1's and the *false* vertices are labeled by 0's. Henceforth, such a hypercube is called the *geometrical representation of f.*

Since the *true* (or *false*) minterms of a Boolean functions are in one-to-one correspondence with the *true* (or *false*) vertices labeled by 1 (or 0) then the minterm expansion form of *f*, the truth table of *f* and the geometrical representation of *f* are equivalent concepts.

Consider the definition of a threshold function realized by structure $[w_1, w_2, ..., w_n;\ \theta]$ and the equation: $\sum_1^n x_i w_i = \theta$. In *n* dimensional space it is the equation of a hyper plane that separates the *true* vertices from the *false* vertices. This is why threshold functions are known as *linearly separable functions.*

The geometrical representation of threshold functions provides a natural way to represent such functions. However, if $n \geq 4$ it is difficult to illustrate. An alternative way of representing threshold functions is by using a *vertex map*. It is a graph that can be obtained by projecting the *true* vertices of the geometrical representation of *f* together with their adjacency edges into 2-dimensional space.

Given an (*n, k*) Boolean-function *f* and its geometrical representation *G*. The vertex map *M* associated with *f* is defined as follows: $V(M) = \{X_1, X_2, \ldots, X_k\}$ denotes the set of vertices of *M*, and *E*(*M*) denotes the set of *M*'s edges. X_i is adjacent to X_j in *M*, written as $(X_i, X_j) \in E(M)$, if and only if X_i and X_j are connected by an edge of the unit hypercube, for $1 \leq i, j \leq k$. In other words, their corresponding minterms differ in exactly one literal. For example, $f = x_1x_2x_3 + x_1x_2\overline{x_3} + \overline{x_1}x_2x_3$ has the following geometrical representation and vertex map:

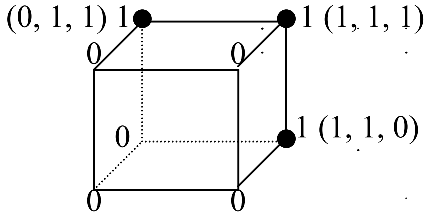

Figure 1: The Geometrical Representation of *f*

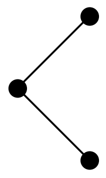
Figure 2: The Vertex Map of *f*

4. Properties of Threshold Functions

4.1. Preserving Operations and Closure Transformations

Given a threshold function *f*. The Boolean function *f'* that can be obtained from *f* by one or combination of the following operations is a threshold function [8]:

(1) Negation of one or more variables;
(2) Permutation of two or more variables; and
(3) Negation of the output function, i.e., $\overline{f}$.

Functions that are equivalent under these three operations are said to be *NPN*-equivalent and belong to the same *NPN*-equivalent class. The *NP*-equivalent class of *f* consists of functions that equivalent to *f* under operations (1) and (2); and *P*-equivalent class is the class of functions equivalent under operation (2) alone.

Vertex maps of Boolean functions *f* and *g* are isomorphic if and only if they belong to the same *NP*-class. Furthermore, if *f* is a threshold function then so is *g*. If *f* is an (*n, k*) threshold-function then its inverse $\overline{f}$ is an (n, 2^n-k) threshold-function.

Besides the preserving operations of threshold functions there are closure transformations as follows: if *f* is a threshold function of *n* variables $x_1, x_2, ..., x_n$, then

(1) $f + x_p$, and
(2) fx_p

are threshold functions, where $1 \le p \le n+1$ [8].

For $p = n+1$, transformation (2) means that an (*n, k*) threshold function *f* can be regarded as an (*n*+1, *k*) threshold-function *g*. Considered geometrically, *g* is an extension of *f* in *n*+1 dimensional space. Note that the vertex maps of *f* and *g* are isomorphic. If *f* is an (*n, k*) threshold-function then fx_{n+1} is an (*n*+1, *k*) threshold-function and $f + x_{n+1}$ is an (n+1, $k+2^n$) threshold-function.

4.2. Complete Monotonicity

Definition 2:
Let *f* and *g* be *n*-variable Boolean functions. If for every *X* satisfying *f*(*X*) = 1 we also have *g*(*X*) = 1 then we say that *f implies g,* denoted by $f(X) \Rightarrow g(X)$. If either $f(X) \Rightarrow g(X)$ or $g(X) \Rightarrow f(X)$ holds then *f* and *g* are said to be comparable. Otherwise they are incomparable. If $f_A \Rightarrow f_{\bar{A}}$ or $f_{\bar{A}} \Rightarrow f_A$ holds for each *k*-assignment *A* for a fixed *k* then *f* is said to be *k-comparable.* If *f* is *k*-comparable for each $1 \le k \le s$ then *f* is said to be *s-monotonic*. If *f* is *n*-monotonic then *f* is said to be *completely monotonic*.

It is cumbersome to check whether or not an *n* variable Boolean function is completely monotonic by examining all complementary *s*-assignments, for $s \le n$. Another way to do this is by looking at the geometrical consequence of the complete monotonicity condition discussed in [13].

Let *AB* and *AC* be two complete assignments. *B* and *C* do not have to be complementary. If *f* is a completely monotonic function then either $f_A \Rightarrow f_{\bar{A}}$ or $f_{\bar{A}} \Rightarrow f_A$ holds for every assignment *A*. Consequently, either ($f_{AB} \Rightarrow f_{\bar{A}B}$ and $f_{AC} \Rightarrow f_{\bar{A}C}$) or ($f_{\bar{A}B} \Rightarrow f_{AB}$ and $f_{\bar{A}C} \Rightarrow f_{AC}$) holds. As a result, if *f* is completely monotonic then it is impossible that *f*(*AB*) = 1, $f(\bar{A}B) = 0$, *f*(*AC*) = 0 and $f(\bar{A}C) = 1$.

Suppose *B* and *C* have common assignment *E*, thus *B* = *DE* and $C = \bar{D}E$ for some differing assignments *D* and $\bar{D}$. Therefore, $AB = ADE$, $\bar{A}B = \bar{A}DE, AC = AE\bar{D}$, and $\bar{A}C = \bar{A}\bar{D}E$. Let *AD* = G and $\bar{A}D = H$. Therefore, $AB=GE$, $AC=\bar{H}E, \bar{A}B = HE, \bar{A}C = \bar{G}E$. As a result, the complete monotonicity condition can be explained as follows:

Lemma 1[13]:
A Boolean function *f* is completely monotonic if and only if there are no *E, G* and *H* such that

$$f(GE) = 1, \ f(\bar{G}E) = 1, f(HE) = 0, \text{ and } f(\bar{H}E) = 0 .$$

If we construct a parallelogram in which $GE, \overline{GE}, HE, \overline{HE}$ are the corner points then $GE, \overline{GE}$ are two diagonally opposing vertices, and $HE, \overline{HE}$ are the other two diagonally opposing vertices. Thus, geometrically, a Boolean function is completely monotonic if and only if there is no parallelogram in which two diagonally opposing vertices are *true* vertices and the other two opposing vertices are *false* vertices. Since minterms, assignments and vertices of a unit hypercube are the same concepts then Lemma 1 can be explained in terms of minterms as follows:

Lemma 2[10]:
Let *f* be an (*n, k*) Boolean-function. Then, *f* is completely monotonic if and only if for every pair of *true* minterms $\mathbf{m}_i$ and $\mathbf{m}_j$ the following condition holds: If **s** is the common subminterms between $\mathbf{m}_i$ and $\mathbf{m}_j$, for every other pair of minterms $\mathbf{m}_k$ and $\mathbf{m}_l$ that have **s** as the common subminterm then at least one of $\mathbf{m}_k$ and $\mathbf{m}_l$ is a *true* minterm of *f*.

The following theorem shows an important property of threshold functions.

Theorem 1 [16]:
Complete monotonicity is a necessary and sufficient condition for a Boolean function to be a threshold function when $n < 9$. For $n = 9$ there are completely monotonic functions which are not threshold functions.

5. Generating Algorithm

The program for generating threshold functions works recursively in terms of the number of minterms. It generates all completely monotonic functions by using Lemma 2 as the checking procedure. Since complete monotonicity is equivalent to being a threshold function for $n \leq 8$, then the program will not generate any non-threshold function of less than nine variables.

If *f* is a threshold function, we can construct new threshold function by adding or multiplying a new variable into *f*. If we are given a set of all *n* variables threshold functions, applying those procedures do not generate all of $n + 1$ variable threshold functions. The following theorems will be used to generate all (*n, k*+1) threshold-functions from (*n, k*) threshold-functions.

Theorem 2 [10]:
Every (*n, k*) threshold function *f*, when $n \leq 8$ and $k \geq 1$, can be decomposed into an (*n, k*-1) threshold-function *g* and a minterm **m** such that $f = g + \mathbf{m}$.

For every (*n, k*) threshold-function *f* there exists an (*n, k*-1) threshold function *g* that can be obtained from *f* by removing one of its minterms, when $n \leq 8$, $k \geq 1$. In other words, every (*n, k*) threshold function can be constructed from an (*n, k*-1) threshold-function by adding a new minterm. Note that adding a function and a minterm can produce a non-threshold function. Consequently, the checking procedure based on Lemma 2 is necessary. The procedure can generate the same threshold function more than once. Therefore, the program contains a procedure to remove redundancy.

Every (*n*, 1) Boolean-function is a threshold function, then every (*n*, 2) threshold function can be generated from an (*n*, 1) threshold-function by adding one minterm. Every (*n*, 2) threshold-function can be generated that way. Once we have the list of all (*n*, 2) threshold-functions then we can generate all (*n*, 3) threshold functions by the same procedure. The theorem guarantees that every

(n,k) threshold-function, for $n \leq 8$ and $1 \prec k \leq 2^n$, can be generated using the procedure. Complementing its inverse, a threshold function having 2^n - k minterms, can generate a threshold function f having k minterms, can be generated more easily. In the program, this can be done by subtracting $\overline{f}$ from {0, 1, 2,…, 2^{n-1}}: f = {0, 1, 2,…, 2^{n-1}} - $\overline{f}$.

Algorithm: Generating all *n* Variable Threshold Functions ($n \leq 8$)

F	: The set of threshold functions generated
M	: The set of all minterms of n variables
$S(f_i)$	: The set of all *true* minterms of f_i
$\overline{S}(f_i)$	: The set of all *false* minterms of f_i
Input	: n (the number of variables).
Output:	all threshold functions of n variables

```
Step 1 : Initialization
    $F \leftarrow \phi$
        $F \leftarrow F \cup \{\phi\}$          (* threshold function having no minterm, zero function *)
        For each minterm of $n$ variable $\mathbf{m}_i$ in $M$          (* Generate all ($n$, 1) threshold functions*)
        $f_i \leftarrow \{ \mathbf{m}_i \}$
        $F \leftarrow F \cup \{ f_i \}$
    End for

Step 2 : Generating Step
    FOR $k$ = 1 TO $2^{n-1}$ DO
        For every ($n$, $k$) threshold function $f$ in $F$
            For every minterm $\mathbf{m}$ in $S(f)$
                For every minterm $\mathbf{m}_i$ in $S(f)$
                    $\mathbf{e} \leftarrow$ differing subminterm between $\mathbf{m}$ and $\mathbf{m}_i$ in $\mathbf{m}$
                    $\bar{\mathbf{e}} \leftarrow$ differing subminterm between $\mathbf{m}$ and $\mathbf{m}_i$ in $\mathbf{m}_i$
                    $\mathbf{s} \leftarrow$ identical subminterm between $\mathbf{m}_i$ and $\mathbf{m}$
                    Completely monotonic $\leftarrow$ true
                    For every minterm in $M$ that contain $\mathbf{s}$, say $\mathbf{as}$,
                        If both $\bar{\mathbf{a}}\mathbf{s} \in \overline{S}(f)$ and $\mathbf{as} \in \overline{S}(f)$
                            Then Completely-monotonic $\leftarrow$ false
                    End for
                End for
            End for
            If Completely-monotonic = true and $f + \mathbf{m}_i \notin F$
                Then $F \leftarrow F \cup \{f + \mathbf{m}_i\}$
        End for
    END

Step 3:          (*   Generating the inverses   * )
    For each  $f \in F$
        $F \leftarrow F \cup \{\overline{f}\}$
```

6. Evaluations

The algorithm works incredibly slow as n getting bigger. Therefore, even though the algorithm valid for $n \leq 8$, the program runs for $n \leq 5$. There are 94,572 threshold functions of five variables and 119 different vertex maps representing those functions, as shown by the following table:

Table 2: The Number of n Variable Threshold Functions and The Vertex Maps

n	The number of threshold functions	The number of vertex maps
0	2	1
1	4	3
2	14	5
3	104	10
4	1882	27
5	94572	119

7. Future Research Topics

The program works very well when $n \leq 5$. When $n \succ 5$ it becomes incredibly slow. Better algorithm is needed to generate threshold functions of five or more variables in reasonable time and space. Furthermore, extending the algorithm given in this paper to make it applicable for $n \succ 8$ we need stronger conditions than the complete monotonicity property alone. Besides that, there are two main questions regarding the vertex maps of threshold functions:

(1) What are the potential uses of vertex maps especially for the characterization of threshold functions?
(2) How can we generate vertex maps of threshold functions of more than five variables?

Acknowledgement

The work and the presentation of this paper sponsored by QUE Project for The Department of Mathematics, The University of Indonesia.

8. References

[1] Akers, S.B.Jr., and B.H.Rutter,"The Use of Threshold Logic in Character Recognition," *Proc. Of the IRE*, August 1964, pp. 931-938.
[2] Bondy, J.A. and U.S. Murty, Graph Theory with Applications, North Holland, New York, N.Y., 1976.
[3] Chow, C.K., "Boolean Functions Realizable with Single Threshold Devices," *Proc. of the IRE*,Vol 49, January 1961, pp. 370-371
[4] Coates, C.L.and P.M.Lewis, "DONUT, a Threshold Gate Computer," *IEEE TEC*, Vol . EC-13, No. 3, June 1964, pp.240-247.
[5] Dearholt, D.W., "On Threshold Logic," PhD Thesis, Dept. of Electrical Engineering, University of Washington, July 1965.
[6] Dertouzos, M.L., *Threshold Logic: A Synthesis Approach*, MIT Press, Cambridge, Mass., 1965
[7] Elgot, C.C., "Truth Functions Realizable by Single Threshold Organ," *SCTLD*,

September 1961, pp. 225-246.

[8] Gabelman, I.J., “Properties and Transformations of Single Threshold Elements Functions,” *IEEE TEC*, Vol. *EC*-13, No. 6, December 1964, pp. 680-684.

[9] Hu, Sze-Tzen, *Threshold Logic*, University California Press, Berkely, 1965.

[10] Kasiyah, “Census and Analysis of threshold Functions,” MSc Thesis, Computer Science Dept., University of Western Ontario, London, Canada, September 1991.

[11] Kaszerman, P., “A Geometric Test-Synthesis Procedure for a Threshold Device,” *Inf. Control*, Vol. 6, No. 1, December 1963, pp. 381-398.

[12] Kaszerman, P., “A ‘Region’ Concept and Its Application to Threshold Logics,”*Inf. Control*, Vol. 8, No. 5, October 1965, pp. 531-551.

[13] Muroga, S., *Threshold Logic and Its Applications*, Wiley-Interscience, New York, N.Y., 1971.

[14] Muroga, S., T. Tsuboi, and C.R.Baugh, “Enumeration of Threshold Function of Eight Variables,” *IEEE TEC*, Vol. C-19, No. 9, September 1970, pp. 818-825.

[15] Sheng, C.L., *Threshold Logic*, Academic Press, New York, N.Y., 1969.

[16] Winder, R.O., “Properties of Threshold Functions,” *IEEE TEC*, Vol. *EC*-14, No.2, April 1965, pp. 252-254.