A Constructive Approach for Threshold Function Identification

MENG-JING LI, YU-CHUAN YEN, and YI-TING LI, National Tsing Hua University, Taiwan
YUNG-CHIH CHEN, National Taiwan University of Science and Technology, Taiwan
CHUN-YAO WANG, National Tsing Hua University, Taiwan

Threshold Function (TF) is a subset of Boolean function that can be represented with a single linear threshold gate (LTG). In the research about threshold logic, the identification of TF is an important task that determines whether a given function is a TF or not. In this article, we propose a sufficient and necessary condition for a function being a TF. With the proposed sufficient and necessary condition, we devise a TF identification algorithm. The experimental results show that the proposed approach saves 80% CPU time for identifying all the 8-input NP-class TFs as compared with the state-of-the-art. Furthermore, the LTGs corresponding to the identified TFs obtained by the proposed approach have smaller weights and threshold values than the state-of-the-art.

CCS Concepts: • **Theory of computation → Design and analysis of algorithms;**

Additional Key Words and Phrases: Threshold logic, linear threshold logic gate, threshold function identification

ACM Reference format:
Meng-Jing Li, Yu-Chuan Yen, Yi-Ting Li, Yung-Chih Chen, and Chun-Yao Wang. 2023. A Constructive Approach for Threshold Function Identification. *ACM Trans. Des. Autom. Electron. Syst.* 28, 5, Article 86 (September 2023), 19 pages.
https://doi.org/10.1145/3606371

1 INTRODUCTION

In recent years, since nanoscale devices [6], such as Resonant Tunneling Diodes [2], Single Electron Transistors [22], Quantum Cellular Automata [16], and Memristors [21], have been rapidly developed and are available for implementing threshold logic gates, researchers pay more attention to the threshold logic than before [7, 8, 11, 14, 24]. Different from the traditional Boolean logic, threshold logic is another representation for Boolean functions. A function that can be represented with a single **linear threshold gate (LTG)** is called a **threshold function (TF)**. However, not all the functions can be represented by a single LTG. For example, a function

This work is supported in part by the National Science and Technology Council (Taiwan) under MOST 109-2221-E-007-082-MY2, MOST 109-2221-E-155-047-MY2, MOST 110-2224-E-007-007, MOST 111-2218-E-007-010, MOST 111-2221-E-007-121, MOST 111-2221-E-011-137-MY3, NSTC 112-2218-E-007-014, and NSTC 112-2425-H-007-002.
Authors' addresses: M.-J. Li, Y.-C. Yen, Y.-T. Li, and C.-Y. Wang, National Tsing Hua University, Taiwan; emails: {amy26656, yuki098711, yitingli.yt}@gmail.com, wcyao@cs.nthu.edu.tw; Y.-C. Chen, National Taiwan University of Science and Technology Taiwan; email: ycchen.ee@mail.ntust.edu.tw.

1084-4309/2023/09-ART86 $15.00
https://doi.org/10.1145/3606371

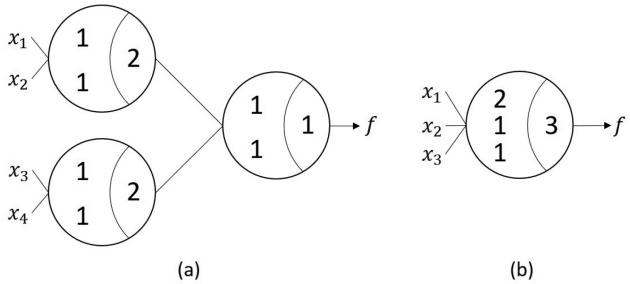

Fig. 1. (a) The threshold network of $f = x_1x_2 + x_3x_4$. (b) The LTG of $f = x_1x_2 + x_1x_3$.

$f = x_1x_2 + x_3x_4$ is a **non-threshold function (non-TF)** and it requires a threshold network with multiple LTGs to represent it as shown in Figure 1(a).

An LTG consists of n binary inputs, $x_1, \ldots, x_n$, with the corresponding weights, $w_1, \ldots, w_n$, and a threshold value T. The output f is evaluated as the following equation:

$$f(x_1, x_2, \ldots, x_n) = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} x_i w_i \geq T \\ 0, & \text{otherwise.} \end{cases}$$

An LTG can also be represented as a weight-threshold vector $[w_1, w_2, \ldots, w_n; T]$. For example, the corresponding LTG of function $f = x_1x_2 + x_1x_3$ is shown in Figure 1(b) and can be expressed as $[2, 1, 1; 3]$.

The studies of threshold logic began from the 1960s. K-asummability [17] was proposed to be a necessary and sufficient condition for a function to be a TF. However, the condition cannot report the corresponding weights and threshold value [4, 5, 23]. The first approach to enumerating TFs was proposed in 1961 [25]. In 1970, a linear programming method for enumerating TFs of 8 input variables was proposed [19]. In general, when a function is identified as a TF, the weights and threshold value have to be determined as well for implementation. Furthermore, TF identification is essential in the synthesis of **threshold logic network (TLN)** [3, 8, 12]. This is because some synthesis techniques rely on TF libraries to generate cost-effective TLNs. While there are currently NP-class TF libraries with up to 8 inputs, our proposed approach for TF identification offers a more efficient method that can accelerate the process. This approach has the potential to identify TFs with more than 8 inputs, such as 9-input TFs.

In this work, we propose another sufficient and necessary condition for a function being a TF. With this condition, we also devise an algorithm to generate a composite inequality system. Since the generated composite inequality system contains fewer inequalities on variables, the algorithm is more efficient to identify all the 8-input NP-class TFs than the state-of-the-art [15].

The main contributions of this article are as follows:

(1) We propose a more efficient sufficient and necessary condition for a function being a TF and construct a *composite* inequality system with this condition.
(2) We propose a new initial weight assignment to accelerate the weight assignment procedure.
(3) The proposed TF identification algorithm saves 80% CPU time and obtains more optimal LTGs with smaller weights and threshold values for identifying all the 8-input NP-class TFs as compared to the state-of-the-art.

The rest of this article is organized as follows: Section 2 introduces the background of threshold logic. Section 3 introduces some previous works related to this topic. Section 4 presents the proposed sufficient and necessary condition, and the TF identification algorithm. Section 5 discusses

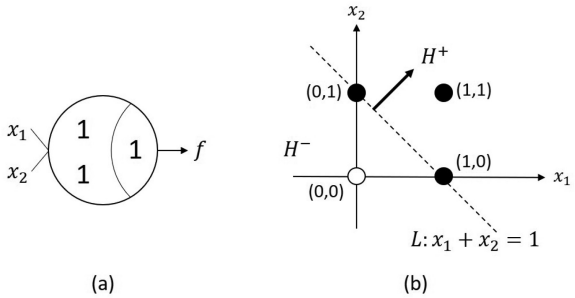

Fig. 2. (a) The LTG of $f = x_1 + x_2$ [11]. (b) Hyperplane and half-spaces of an OR gate [11].

the time complexity of our approach. Section 6 shows the experimental results. Section 7 concludes this article and discusses the future work.

2 PRELIMINARIES

In this section, we introduce some fundamentals about threshold logic.

2.1 Backgrounds

The *on-set* (*off-set*) of a Boolean function f is a set of minterms such that $f = 1$ ($f = 0$).

An *NP-class* is a group of Boolean functions that can be derived from each other by permuting the input variables or taking their complements. For example, functions such as $f = x_1 + x_2x_3$, $g = x_3 + x_1x_2$, and $h = x_1' + x_2x_3$ are in the same NP-class.

An n-input function $f(x_1, x_2, \ldots, x_i, \ldots, x_n)$ is said to be *positive unate* in variable x_i if and only if for all possible values in variable x_j, $j \neq i$,

$$f(x_1, x_2, \ldots, x_{i-1}, x_i = 1, x_{i+1}, \ldots, x_n) \geq f(x_1, x_2, \ldots, x_{i-1}, x_i = 0, x_{i+1}, \ldots, x_n).$$

If f is positive unate for each variable, then f is said to be a *unate function* or the function has the unateness property.

2.2 Hyperplane and Half-space

A TF can be represented as a hyperplane H in an n-dimensional space, i.e., $H : \sum_{i=1}^{n} x_i w_i = T$. The hyperplane separates the on-set minterms and off-set minterms of TF in two half-spaces. This is the linear separability of a TF. The on-set minterms are located in the positive half-space $H^+ : \sum_{i=1}^{n} x_i w_i \geq T$, while the off-set minterms are located in the negative half-space $H^- : \sum_{i=1}^{n} x_i w_i < T$.

For example, as the LTG of function $f = x_1 + x_2$ shown in Figure 2(a), there exists a hyperplane $L : x_1 + x_2 = 1$ separating the on-set minterms, x_1x_2, $x_1'x_2$, x_1x_2', from the off-set minterm, $x_1'x_2'$, as shown in Figure 2(b). Therefore, a TF can be interpreted as a function having a hyperplane separating two half-spaces.

2.3 Chow's Parameter

Chow's parameter is used to determine the variable ordering of a TF in the previous work [15, 20]. Given a function $f(x_1, x_2, \ldots, x_n)$, the Chow's parameter $P(f)$ is a vector defined in the following equation:

$$P(f) = (p_1(f), p_2(f), \ldots, p_n(f); p_0(f)),$$

where $p_i(f), 1 \leq i \leq n$, is the number of minterms in the on-set for which $x_i = 1$, and $p_0(f)$ is the number of minterms in the on-set of f. The elements in Chow's parameters about two variables induce the relationship between the corresponding weights of these two variables [20]. Therefore, the input variable with a larger $p_i(f)$ has a larger weight in its LTG.

2.4 Shannon's Expansion

Shannon's expansion decomposes a function $f(x_1, x_2, \ldots, x_n)$ in the n-dimension Boolean space B^n into two cofactor functions as the following equation:

$$f(x_1, x_2, \ldots, x_n) = x_i^{'} \cdot f(x_i = 0) + x_i \cdot f(x_i = 1)$$

for $1 \leq i \leq n$, where $f(x_i = 0)$ and $f(x_i = 1)$ are cofactor functions with respect to x_i.

2.5 Equivalence Classes

Muroga et al. [18] selected a representative function $f(x_1, x_2, \ldots, x_n)$ for every NP-class of TFs with the following criteria:

(1) $\forall i \in \{1, \ldots, n\}, f(x_i = 0) \leq f(x_i = 1)$,
(2) $x_1 \geq x_2 \geq \cdots \geq x_n$, where $x_i \geq x_j$ denotes $f(x_i = 0, x_j = 1) \leq f(x_i = 1, x_j = 0)$.

We use these criteria to select representative TFs for all NP-classes. The first criterion indicates that the function is positive unate in every variable [9]. Without loss of generality, we assume that the weights and threshold value are positive, and thus the TF is positive unate in every variable. This is because the negative weights and negative threshold value in an LTG can be converted to positive ones by applying the positive-negative weight transformation method [17]. The second criterion indicates $w_1 \geq w_2 \geq \cdots \geq w_n$. In this work, we assume that the weights in the weight vector are sorted in a descending order.

3 RELATED WORKS

In this section, we review some related works about threshold function identification.

The TF identification methods can be divided into two classes: (1) **integer linear programming (ILP)**-based methods and (2) non-ILP-based methods. The ILP-based method [10, 19, 26] formulated the TF identification problem as an ILP problem and expolited ILP solvers to determine the weight-threshold vector of the function.

The non-ILP-based methods [15, 20] apply heuristics to assign weights and threshold value to a function. Neutzling et al. [20] propose a method to generate an inequality system from **irredundant sum-of-products (ISOPs)** of a function. Then, a weight assignment procedure adjusts the weights and checks the consistency of the inequality system. If consistent assignments about the weights are found, then a TF is identified and the threshold value is computed; otherwise the function is an undetermined function. This method is able to identify all the TFs with up to six variables.

The procedures of inequality system generation and weight assignments are then improved by Liu et al. [15]. First, Liu et al. simplify the inequality system by removing redundancies. Then, they propose a new weight assignment algorithm that searches weights from the solution space more comprehensively. The new algorithm identifies all the TFs with up to eight variables.

Unateness property is a necessary condition for being TFs, and the studies of References [15] and [20] use this property to accelerate the identification procedure. When the function-under-identification is a non-TF but with the unateness property, the weight assignment procedure would be very time-consuming and in vain. To improve the TF identification algorithm, Lin et al. propose a new and more effective necessary condition that detects more non-TFs with unateness property [13].

This new necessary condition can be seamlessly integrated into the identification algorithm in Reference [15].

In Section 3.1, we introduce the sufficient and necessary condition for a function being a TF [7, 17] and explain their weakness as we also propose a sufficient and necessary condition for a function being a TF. In Section 3.2, we introduce the threshold function identification method proposed in the state-of-the-art [15].

3.1 Summable Theorem

Summable Theorem [17] is a sufficient and necessary condition for a function being a TF. A function f is said to be k-summable if and only if for some k, $2 \leq k$, there exist a subset of the on-set of f, denoted as A, and a subset of the off-set of f, denoted as B, such that $A^{(j)}$ is an on-set minterm $\in A$ and $B^{(j)}$ is an off-set minterm $\in B$, and

$$A^{(1)} + A^{(2)} + \cdots + A^{(k)} = B^{(1)} + B^{(2)} + \cdots + B^{(k)},$$

where + denotes the component-wise addition of vectors, provided that repetition of vectors in the sets A and B are permitted. The Summable Theorem further states that a function f is a TF if and only if f is asummable. Given the number of on-set (off-set) minterms s (t) of a function f, according to definition of k-summable, the time complexity of checking k-summable for the function f is $O(s^k * t^k)$.

Hsu et al. [7] then propose a *Semi-critical Summable Theorem* as a new sufficient and necessary condition of threshold function, which reduces the searching space for Summable Theorem [17] from on-set vectors and off-set vectors to *ONCV*s and *OFFCV*s.

Despite their benefits, these approaches have practical limitations when it comes to identifying TFs. One major drawback is their inability to determine weight assignments. Additionally, the approaches' permission of repetition of vectors in sets A and B can lead to an unbounded checking process, which is unsuitable for practical use.

3.2 Review of the State-of-the-art

The state-of-the-art approach first checks if a function is unate or not. If it is not unate, then it is classified as a non-TF. However, if the function is unate, then the state-of-the-art method proceeds to build the inequality system. Once the inequality system is established, the approach identifies the functions that violate **Variable Weight Ordering (VWO)** as non-TFs. After the function passes both of these necessary conditions, it enters the weight assignment procedure, where the weight gradually increases. If the maximum weight in the assignment exceeds the theoretical weight upper bound mentioned in Reference [17], the function is then identified as an undetermined function.

We use an example to demonstrate the TF identification algorithm [15]. Given an 8-input function $f = x_1x_2x_3x_4x_5 + x_1x_2x_3x_4x_6 + x_1x_2x_3x_4x_7 + x_1x_2x_3x_4x_8 + x_1x_2x_3x_5x_6 + x_1x_2x_3x_5x_7 + x_1x_2x_3x_5x_8 + x_1x_2x_3x_6x_7 + x_1x_2x_3x_6x_8 + x_1x_2x_4x_5x_6 + x_1x_2x_4x_5x_7x_8 + x_1x_2x_4x_6x_7x_8 + x_1x_2x_5x_6x_7x_8 + x_1x_3x_4x_5x_6 + x_1x_3x_4x_6x_7x_8 + x_1x_3x_5x_6x_7x_8 + x_2x_3x_4x_5x_6x_7 + x_2x_3x_4x_5x_6x_8 + x_2x_3x_4x_5x_7x_8 + x_2x_3x_4x_6x_7x_8 + x_2x_3x_5x_6x_7x_8$, the VWO of the function is determined according to the Chow's parameter, i.e., $x_1 > x_2 = x_3 > x_4 = x_5 = x_6 > x_7 = x_8$.

The next step is to generate inequalities from the ISOPs form of f and f'. The greater side of the inequalities is obtained from the on-set of f, i.e., the product terms of f, and its lesser side is obtained from the off-set of f, i.e., the product terms of f'. There are some redundant product terms in the on-set and off-set. Hence, it is not necessary to compare all the product terms in the on-set with all the product terms in the off-set. For a product term in the on-set, it is redundant if it has a *larger* weight summation than other product terms. Therefore, the don't care bits of product

Greater side		*Lesser side*
		$w_1+w_2+w_3+w_4$
$w_1+w_2+w_3+w_4+w_7$		$w_1+w_2+w_3+w_7+w_8$
$w_1+w_2+w_4+w_5+w_6$	$\geqq T >$	$w_1+w_2+w_4+w_5+w_7$
$w_2+w_3+w_4+w_5+w_7+w_8$		$w_1+w_4+w_5+w_6+w_7+w_8$
		$w_2+w_3+w_4+w_5+w_6$

Fig. 3. List of irredundant weight summation in the greater side and lesser side of f.

#	Inequality		
1	$w_1+w_2+w_3+w_4+w_7$	$>$	$w_1+w_2+w_3+w_4$
2	$\boldsymbol{w_1+w_2+w_3}+w_4+\boldsymbol{w_7}$	$>$	$\boldsymbol{w_1+w_2+w_3+w_7}+w_8$
3	$w_1+w_2+w_3+w_4+w_7$	$>$	$w_1+w_2+w_4+w_5+w_7$
4	$w_1+w_2+w_3+w_4+w_7$	$>$	$w_1+w_4+w_5+w_6+w_7+w_8$
5	$w_1+w_2+w_3+w_4+w_7$	$>$	$w_2+w_3+w_4+w_5+w_6$
6	$w_1+w_2+w_4+w_5+w_6$	$>$	$w_1+w_2+w_3+w_4$
7	$w_1+w_2+w_4+w_5+w_6$	$>$	$w_1+w_2+w_3+w_7+w_8$
8	$w_1+w_2+w_4+w_5+w_6$	$>$	$w_1+w_2+w_4+w_5+w_7$
9	$w_1+w_2+w_4+w_5+w_6$	$>$	$w_1+w_4+w_5+w_6+w_7+w_8$
10	$w_1+w_2+w_4+w_5+w_6$	$>$	$w_2+w_3+w_4+w_5+w_6$
11	$w_2+w_3+w_4+w_5+w_7+w_8$	$>$	$w_1+w_2+w_3+w_4$
12	$w_2+w_3+w_4+w_5+w_7+w_8$	$>$	$w_1+w_2+w_3+w_7+w_8$
13	$w_2+w_3+w_4+w_5+w_7+w_8$	$>$	$w_1+w_2+w_4+w_5+w_7$
14	$w_2+w_3+w_4+w_5+w_7+w_8$	$>$	$w_1+w_4+w_5+w_6+w_7+w_8$
15	$w_2+w_3+w_4+w_5+w_7+w_8$	$>$	$w_2+w_3+w_4+w_5+w_6$

Fig. 4. Original inequality system of f.

terms in the on-set can be ignored in the generation of greater side of inequality. For example, the product term $x_1x_2x_3x_4x_5$ is corresponding to $w_1 + w_2 + w_3 + w_4 + w_5$ where don't care bits x_6, x_7, x_8 are ignored. Furthermore, some products terms in the on-set can be removed as referring to the VWO. For example, product terms $x_1x_2x_3x_4x_5$ and $x_1x_2x_3x_4x_7$ are both in the on-set of f. Since $x_5 = x_6 > x_7$ in the VWO, the weight summation of $x_1x_2x_3x_4x_5$, i.e., $w_1 + w_2 + w_3 + w_4 + w_5$, is larger than that of $x_1x_2x_3x_4x_7$, i.e., $w_1 + w_2 + w_3 + w_4 + w_7$. Hence, $x_1x_2x_3x_4x_5$ is redundant and can be removed in the generation of the inequalities.

Similarly, the don't care bits of product terms in the off-set have to be kept while non-don't care bits of product terms are ignored. For example, $x_1'x_2'$ corresponds to $w_3 + w_4 + w_5 + w_6 + w_7 + w_8$. A product term in the off-set is redundant if it has a *smaller* weight summation than other product terms. The weight summation of $x_1'x_5'x_6'$, i.e., $w_2 + w_3 + w_4 + w_7 + w_8$, is smaller than that of product terms $x_1'x_7'x_8'$, i.e., $w_2 + w_3 + w_4 + w_5 + w_6$, as referring to the VWO. Hence, $x_1'x_5'x_6'$ is redundant. The irredundant weight summations in the greater side and the lesser side of this example are shown in Figure 3. Then, the weight summations in the greater side are paired with the weight summations in the lesser side to generate the inequality system as shown in Figure 4.

The next step is to remove redundant inequalities from the inequality system of Figure 4. When the weight summation in the greater side of an inequality is larger than that in the lesser side under the VWO, this inequality is definitely satisfiable and does not need to check its consistency. For example, the 2nd inequality "$\boldsymbol{w_1 + w_2 + w_3} + w_4 + \boldsymbol{w_7} > \boldsymbol{w_1 + w_2 + w_3 + w_7} + w_8$" in Figure 4 is a redundant inequality, since $x_4 > x_8$ in the VWO. The algorithm removes the redundant inequalities and constructs the irredundant inequality system as shown in Figure 5.

#	Inequality		
1	$w_1+w_2+w_3+w_4+w_7$	>	$w_1+w_4+w_5+w_6+w_7+w_8$
2	$w_1+w_2+w_3+w_4+w_7$	>	$w_2+w_3+w_4+w_5+w_6$
3	$w_1+w_2+w_4+w_5+w_6$	>	$w_1+w_2+w_3+w_4$
4	$w_1+w_2+w_4+w_5+w_6$	>	$w_1+w_2+w_3+w_7+w_8$
5	$w_1+w_2+w_4+w_5+w_6$	>	$w_1+w_4+w_5+w_6+w_7+w_8$
6	$w_2+w_3+w_4+w_5+w_7+w_8$	>	$w_1+w_2+w_3+w_4$
7	$w_2+w_3+w_4+w_5+w_7+w_8$	>	$w_1+w_2+w_3+w_7+w_8$
8	$w_2+w_3+w_4+w_5+w_7+w_8$	>	$w_1+w_2+w_4+w_5+w_7$
9	$w_2+w_3+w_4+w_5+w_7+w_8$	>	$w_1+w_4+w_5+w_6+w_7+w_8$
10	$w_2+w_3+w_4+w_5+w_7+w_8$	>	$w_2+w_3+w_4+w_5+w_6$

Fig. 5. Irredundant inequality system of f.

#	Inequality		
1	B+B	>	C+C+D
2	A+D	>	C+C
3	C+C	>	B
4	C+C+C	>	B+D+D
5	B	>	D+D
6	C+D+D	>	A
7	C+C	>	A
8	B+D	>	A
9	B+B	>	A+C
10	D+D	>	C

Fig. 6. Irredundant inequality system of f with reformatted symbols.

The algorithm then reformats the same weight w_i with the same symbol, i.e., $A = w_1, B = w_2 = w_3, C = w_4 = w_5 = w_6, D = w_7 = w_8$. To simplify the inequality system, the algorithm removes the symbols appearing in both sides among all the inequalities. The updated inequality system is shown in Figure 6.

The weight assignment is the other important task in the TF identification algorithm. The initial weight assignments are the least positive integers, i.e., $A = 4, B = 3, C = 2, D = 1$. The algorithm keeps increasing the weight assignments with intent to satisfy more inequalities and finally obtains the correct weight assignment $(8, 7, 5, 3)$ satisfying all the inequalities. Note that the weight assignment procedure does not exhaustively search every possible assignment. Instead, it relies on false inequalities to determine the next assignment. The d value, $d(w_i)$ = (occurrence number of w_i on the greater side of false inequalities) - (occurrence number of w_i on the lesser side of false inequalities), can be calculated for each weight w_i. The algorithm then only adjusts weights with a positive d value. The weight assignment procedure is shown as Figure 7. Under the initial weight vector $(4, 3, 2, 1)$, we have $d(A) = -4$, $d(B) = 3$, $d(C) = 1$, $d(D) = 5$. Thus, we can adjust the weight vector $(4, 3, 2, 1)$ by increasing B, C, and D individually. By increasing B, the weight vector becomes $(5, 4, 2, 1)$; by increasing C, the weight vector becomes $(5, 4, 3, 1)$; and by increasing D, the weight vector becomes $(5, 4, 3, 2)$. Note that all of the weights are distinct, and the larger weight is increased if necessary when the smaller weight is increased. The algorithm uses breadth-first search to obtain a successful weight vector $(8, 7, 5, 3)$, meaning that the one with the smallest weight summation is selected.

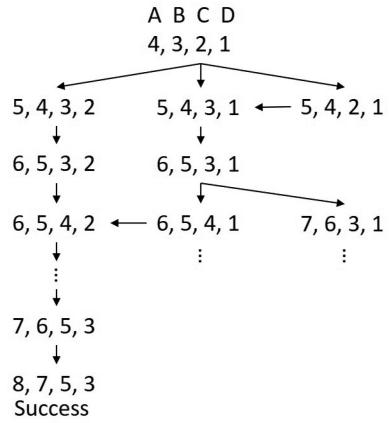

Fig. 7. Weight assignment procedure of f.

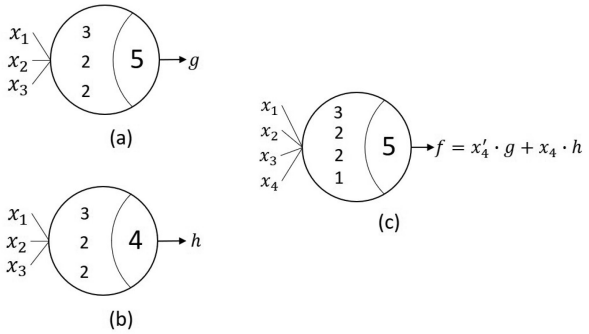

Fig. 8. (a) The LTG of $g = x_1x_2 + x_1x_3$. (b) The LTG of $h = x_1x_2 + x_1x_3 + x_2x_3$. (c) The LTG of $f = x_1x_2 + x_1x_3 + x_2x_3x_4$, which is from the combination of g and h.

The last step is to compute the threshold value after having a correct weight assignment. The threshold value is computed by adding 1 to the largest weight summation in the off-set. The weight summations in the lesser side of Figure 3 under the weight assignment $(8, 7, 5, 3)$ are $27, 28, 28, 29, 29$, respectively. Hence, the threshold value is computed as $29 + 1 = 30$. Finally, the function f is successfully identified as a TF with the corresponding LTG $[8, 7, 7, 5, 5, 5, 3, 3; 30]$.

4 THRESHOLD FUNCTION IDENTIFICATION

In this section, we present the proposed sufficient and necessary condition for a function being a TF. We also propose a new initial weight assignment that can accelerate the weight assignment procedure. Next, we introduce the flow of the proposed TF identification algorithm in detail.

4.1 Sufficient and Necessary Condition for Function Being TF

In this subsection, we present the proposed sufficient and necessary condition for a function being a TF. We first use an example to explain the intention of this condition stated in Theorem 1. Given two 3-input TFs $g(x_1, x_2, x_3) = x_1x_2 + x_1x_3$ and $h(x_1, x_2, x_3) = x_1x_2 + x_1x_3 + x_2x_3$, TF g can be expressed as $[3, 2, 2; 5]$ and TF h can be expressed as $[3, 2, 2; 4]$ as shown in Figures 8(a) and 8(b). Both TFs have the same weight vector $[3, 2, 2]$, and the threshold value of g, $T_g = 5$, is larger than the threshold value of h, $T_h = 4$. Hence, a composite 4-input function $f(x_1, x_2, x_3, x_4) = x_4^{'} \cdot g +$

$x_4 \cdot h = x_1x_2 + x_1x_3 + x_2x_3x_4$ is also a TF and can be expressed as $[3, 2, 2, 1; 5]$, where the 4th weight $= T_g - T_h = 5 - 4 = 1$, and $T_f = 5 = T_g$ as shown in Figure 8(c).

Theorem 1. *An $(n+1)$-input function $f(x_1, \ldots, x_n, x_{n+1})$ is a TF with the weight-threshold vector $[w_1, \ldots, w_n, w_{n+1}; T_f]$ if and only if there exist a weight vector $[w_1, \ldots, w_n]$, threshold values $T_{f(x_{n+1}=0)} = T_f$, and $T_{f(x_{n+1}=1)} = T_f - w_{n+1}$, such that*

(1) *the n-input cofactor function $f(x_1, \ldots, x_n, x_{n+1} = 0)$ is a TF with the weight-threshold vector $[w_1, \ldots, w_n ; T_{f(x_{n+1}=0)}]$;*
(2) *the n-input cofactor function $f(x_1, \ldots, x_n, x_{n+1} = 1)$ is a TF with the weight-threshold vector $[w_1, \ldots, w_n; T_{f(x_{n+1}=1)}]$;*
(3) *$w_n \geq (T_{f(x_{n+1}=0)} - T_{f(x_{n+1}=1)}) > 0$;*

where w_i is the corresponding weight of input x_i.

Proof. ($\Longleftarrow$) Given the premises that two TFs $[w_1, \ldots, w_n; T_{f(x_{n+1}=0)}]$, $[w_1, \ldots, w_n; T_{f(x_{n+1}=1)}]$, and $w_n \geq (T_{f(x_{n+1}=0)} - T_{f(x_{n+1}=1)}) > 0$. To obtain the LTG of composite $(n+1)$-input function $f = x'_{n+1} \cdot f(x_{n+1} = 0) + x_{n+1} \cdot f(x_{n+1} = 1)$, we keep the weights $w_1, \ldots, w_n$ from the original n-input TFs and compute the additional weight w_{n+1} and threshold value T_f. We discuss this proof in two parts. For the first part, since the input $x_{n+1} = 0$ contributes 0 ($x_{n+1} \cdot w_{n+1} = 0$) to the weight summation, the threshold value T_f should be equal to $T_{f(x_{n+1}=0)}$. Similarly, for the second part, the input $x_{n+1} = 1$ contributes w_{n+1} ($x_{n+1} \cdot w_{n+1} = w_{n+1}$) to the weight summation such that T_f should be equal to $w_{n+1} + T_{f(x_{n+1}=1)}$. As combining both parts, we have $T_f = T_{f(x_{n+1}=0)} = w_{n+1} + T_{f(x_{n+1}=1)}$, i.e., $w_{n+1} = T_{f(x_{n+1}=0)} - T_{f(x_{n+1}=1)}$. Since $w_n \geq T_{f(x_{n+1}=0)} - T_{f(x_{n+1}=1)}$ is one of the premise, we have $w_n \geq w_{n+1}$, which meets the VWO mentioned in Section 2.5. As a result, the LTG of $(n+1)$-input function f is obtained with the weight-threshold vector $[w_1, \ldots, w_n, T_{f(x_{n+1}=0)} - T_{f(x_{n+1}=1)}; T_{f(x_{n+1}=0)}]$. Since the weight-threshold vector of $(n+1)$-input function f has been determined, by the definition of TF, f is a TF. This condition reveals that two n-input TFs with the same weight vector can be combined to construct an $(n+1)$-input TF.

($\Longrightarrow$) Given the weight-threshold vector $[w_1, \ldots, w_n, w_{n+1}; T_f]$ of an $(n+1)$-input TF $f(x_1, \ldots, x_n, x_{n+1})$. To obtain the corresponding LTG for the cofactor function $f(x_1, \ldots, x_n, x_{n+1} = 0)$, we remove the input x_{n+1} and the weight w_{n+1} from the original LTG, and then update the threshold value. Since the input $x_{n+1} = 0$ contributes 0 to the weight summation, the updated threshold value is still T_f, i.e., $T_{f(x_{n+1}=0)} = T_f$. Similarly, the resultant LTG of cofactor function $f(x_1, \ldots, x_n, x_{n+1} = 1)$ is obtained by removing x_{n+1} and w_{n+1} and updating the threshold value $T_{f(x_{n+1}=1)}$ as $(T_f - w_{n+1})$, i.e., $T_{f(x_{n+1}=1)} = T_f - w_{n+1}$. Since $w_1, \ldots, w_n$ are not changed, $f(x_1, \ldots, x_n, x_{n+1} = 0)$ and $f(x_1, \ldots, x_n, x_{n+1} = 1)$ can be expressed with the same weight vector $[w_1, \ldots, w_n]$. According to the VWO in Section 2.5, $w_n \geq w_{n+1}$ and $w_{n+1} > 0$. Since we have $T_{f(x_{n+1}=0)} = T_f$, and $T_{f(x_{n+1}=1)} = T_f - w_{n+1}$ from the threshold value update, we obtain $w_{n+1} = T_f - T_{f(x_{n+1}=1)} = T_{f(x_{n+1}=0)} - T_{f(x_{n+1}=1)}$. As a result, $w_n \geq w_{n+1} = T_{f(x_{n+1}=0)} - T_{f(x_{n+1}=1)} > 0$. In summary, two n-input TFs are obtained as (1) and (2), and their threshold values $T_{f(x_{n+1}=0)}$ and $T_{f(x_{n+1}=1)}$ satisfy (3). This condition reveals that for any $(n+1)$-input TF f, there exist two n-input TFs that can be derived from f. □

4.2 Composite Inequality System Generation

According to Theorem 1 in Section 4.1, we have known that the weight-threshold vector of an $(n+1)$-input TF $f(x_1, \ldots, x_n, x_{n+1})$ can be obtained from the weight-threshold vectors of its n-input cofactor functions $f(x_1, \ldots, x_n, x_{n+1} = 0)$ and $f(x_1, \ldots, x_n, x_{n+1} = 1)$ by composition. The weights of the first n variables are equal to the weights of $f(x_1, \ldots, x_n, x_{n+1} = 0)$ and $f(x_1, \ldots, x_n, x_{n+1} = 1)$, i.e., $[w_1, w_2, \ldots, w_n]$. Therefore, an important issue is how to obtain two

#	Inequality		
1	C+C	>	B
2	A+D	>	C+C
3	C+C+D	>	A
4	B+C	>	A

(a)

#	Inequality		
1	B+B	>	C+C+D
2	B+D	>	A
3	C+C	>	A
4	B+B	>	A+C

(b)

Fig. 9. (a) The irredundant inequality system of $f(x_8 = 0)$. (b) The irredundant inequality system of $f(x_8 = 1)$.

#	Inequality			
1	**C+C**	**>**	**B**	$\boldsymbol{f(x_1, \ldots, x_7, x_8 = 0)}$
2	A+D	>	C+C	
3	**C+C+D**	**>**	**A**	
4	**B+C**	**>**	**A**	
5	B+B	>	C+C+D	$\boldsymbol{f(x_1, \ldots, x_7, x_8 = 1)}$
6	B+D	>	A	
7	C+C	>	A	
8	B+B	>	A+C	

Fig. 10. The composite inequality system of $f(x_1, \ldots, x_8)$.

n-input TFs having the same weight vector of $[w_1, w_2, \ldots, w_n]$. Since there might exist more than one hyperplane that separates the on-set and off-set of a TF, a TF can be represented in different weight-threshold vectors. For example, given a 3-input TF $g(x_1, x_2, x_3) = x_1x_2+x_1x_3$, both $[2, 1, 1; 3]$ and $[3, 2, 2; 5]$ represent g. Thus, in this subsection, we will present an efficient method generating the inequality system for finding a *common weight vector* between $f(x_1, \ldots, x_n, x_{n+1} = 0)$ and $f(x_1, \ldots, x_n, x_{n+1} = 1)$.

We first generate the irredundant inequality systems of n-input cofactor functions $f(x_1, \ldots, x_n, x_{n+1} = 0)$ and $f(x_1, \ldots, x_n, x_{n+1} = 1)$. As Reference [15] did, we adopt that when two input variables x_i and x_j have the same VWO in the $(n+1)$-input function f, their corresponding weights w_i and w_j in the common weight vector will be equal, where $1 \leq i, j \leq n$. Therefore, we use the corresponding VWO of the function f in generating the irredundant inequality systems with weights $w_1, \ldots, w_n$. For example, for the same $(7 + 1)$-input function $f(x_1, \ldots, x_8)$ in Section 3.2, we obtain the 7-input cofactor functions $f(x_1, \ldots, x_7, x_8 = 0)$ and $f(x_1, \ldots, x_7, x_8 = 1)$ with respect to the input variable x_8. Their irredundant inequality systems are generated as shown in Figures 9(a) and 9(b). Note that the same weights w_i are replaced with the same symbol according to the VWO, i.e., $A = w_1, B = w_2 = w_3, C = w_4 = w_5 = w_6, D = w_7$. To find a weight vector that simultaneously satisfies the both inequality systems, we merge the inequality systems of $f(x_1, \ldots, x_7, x_8 = 0)$ and $f(x_1, \ldots, x_7, x_8 = 1)$. The composite inequality system is shown in Figure 10.

We observe that there might exist redundant inequalities in the composite inequality system. For two inequalities i and j, when the weight summation of i in the greater side is larger than that of j, and the weight summations of i and j in the lesser sides are equal under the VWO, the inequality i is redundant and can be removed. For example, in Figure 10, the 3rd and the 7th inequalities meet this requirement. Hence, the 3rd inequality can be removed. Similarly, the 4th inequality can be removed as well when comparing with the 6th inequality. However, when the weight summations of i and j in the greater sides are equal, and the weight summation of i in the lesser side is smaller than that of j under the VWO, the inequality i is redundant and can be removed. For example, in Figure 10, the 1st and the 7th inequalities meet this requirement. Hence, the 1st inequality can

#	Inequality		
1	A+D	>	C+C
2	B+B	>	C+C+D
3	B+D	>	A
4	C+C	>	A
5	B+B	>	A+C

Fig. 11. The simplified composite inequality system of $f(x_1, \ldots, x_8)$.

Greater side		*Lesser side*
$w_1+w_2+w_3+w_4+w_7$		$w_1+w_2+w_3+w_4$
$w_1+w_2+w_4+w_5+w_6$	$\geqq T >$	$w_1+w_2+w_4+w_5+w_7$
$w_2+w_3+w_4+w_5+w_6+w_7$		$\boldsymbol{w_2+w_3+w_4+w_5+w_6}$

Fig. 12. List of irredundant weight summation in the greater side and lesser side of $f(x_1, \ldots, x_7, x_8 = 0)$.

be removed. The simplified composite inequality system is constructed as shown in Figure 11. Note that, since the inequality systems of cofactor functions themselves are irredundant, we just compare an inequality i in the inequality system of $f(x_1, \ldots, x_n, x_{n+1} = 0)$ with an inequality j in the inequality system of $f(x_1, \ldots, x_n, x_{n+1} = 1)$ for efficiency elevation.

4.3 Weight Assignment and Threshold Value Computation

We can use the weight assignment algorithm from Reference [15] to compute the weight vector $[w_1, \ldots, w_n]$ satisfying all the inequalities in the simplified composite inequality system, which is the common weight vector for the two n-input cofactor functions. Next, we need to compute the last weight w_{n+1} and the threshold value T_f of the $(n+1)$-input function $f(x_1, \ldots, x_n, x_{n+1})$.

We first compute the threshold value of n-input cofactor functions, $f(x_1, \ldots, x_n, x_{n+1} = 0)$ and $f(x_1, \ldots, x_n, x_{n+1} = 1)$. The weight summation in the greater side is strictly larger than that in the lesser side because of the linear separability of a TF. According to Theorem 1, the last weight has to satisfy $w_{n+1} = T_{f(x_{n+1}=0)} - T_{f(x_{n+1}=1)}$ and the threshold value should be $T_f = T_{f(x_{n+1}=0)}$. We can obtain the least $T_{f(x_{n+1}=0)}$ by adding 1 to the largest weight summation in the lesser side of $f(x_1, \ldots, x_n, x_{n+1} = 0)$. Similarly, we can obtain the largest $T_{f(x_{n+1}=1)}$ by reusing the least weight summation in the greater side of $f(x_1, \ldots, x_n, x_{n+1} = 1)$. Thus, we will obtain the smallest w_{n+1} and threshold value T_f due to $w_{n+1} = T_{f(x_{n+1}=0)} - T_{f(x_{n+1}=1)}$ and $T_f = T_{f(x_{n+1}=0)}$.

For example, in Figure 11, assume that the initial weight assignment, $A = 4, B = 3, C = 2, D = 1$, is represented as $(4, 3, 2, 1)$, with corresponding d values of -3, 3, 1, and 1, respectively. We then adjust the weights by increasing B, C, and D individually until we obtain a new weight vector of $(5, 4, 3, 2)$, where all weights are distinct. After a few iterations, the weight assignment algorithm returns a common weight vector $(7, 6, 4, 2)$ satisfying all the inequalities in Figure 11. The weight summations in the greater side and lesser side of $f(x_1, \ldots, x_7, x_8 = 0)$ and $f(x_1, \ldots, x_7, x_8 = 1)$ are shown in Figures 12 and 13, respectively. The largest weight summation in the lesser side of $f(x_1, \ldots, x_7, x_8 = 0)$ is $w_2 + w_3 + w_4 + w_5 + w_6 = 6 + 6 + 4 + 4 + 4 = 24$, and the threshold value $T_{f(x_8=0)} = 24 + 1 = 25$ as suggested. However, the least weight summation in the greater side of $f(x_1, \ldots, x_7, x_8 = 1)$ is $w_2 + w_3 + w_4 + w_5 + w_7 = 6 + 6 + 4 + 4 + 2 = 22$, and the threshold value $T_{f(x_8=1)} = 22$ as suggested.

However, according to (3) in Theorem 1, we have the requirement that $w_n \geq w_{n+1} = T_{f(x_{n+1}=0)} - T_{f(x_{n+1}=1)} > 0$. The computed weight vector $(7, 6, 4, 2)$ does not satisfy (3) in Theorem 1 due to

Greater side		*Lesser side*
$w_1+w_2+w_3+w_4$	$\geqq T >$	$w_1+w_2+w_3+w_7$
$\boldsymbol{w_2+w_3+w_4+w_5+w_7}$		$w_1+w_2+w_4+w_5$
		$w_1+w_4+w_5+w_6+w_7$

Fig. 13. List of irredundant weight summation in the greater side and lesser side of $f(x_1, \ldots, x_7, x_8 = 1)$.

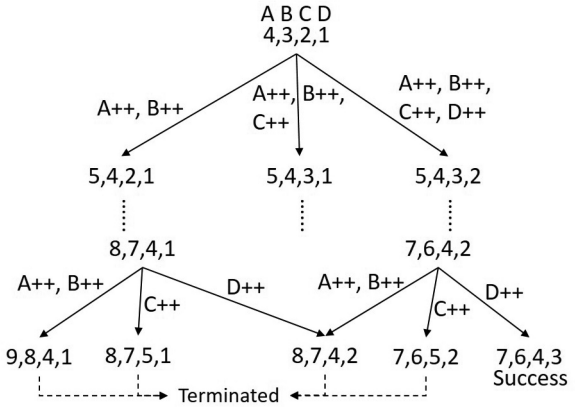

Fig. 14. The weight assignment procedure for $f(x_1, \ldots, x_8)$.

$w_7 = 2 < w_8 = T_{f(x_8=0)} - T_{f(x_8=1)} = 25 - 22 = 3$. As a result, we need to adjust the weight vector $(7, 6, 4, 2)$ by individually increasing the smaller weights. Note that the new weight vector still has to meet the composite inequality system. The weight assignment procedure is summarized in Figure 14. At last, the weight vector $(7, 6, 4, 3)$ is obtained, as it satisfies (3) in Theorem 1, and $T_{f(x_8=0)} = 25$, $T_{f(x_8=1)} = 23$. Also, $w_7 = 3 > w_8 = T_{f(x_8=0)} - T_{f(x_8=1)} = 25 - 23 = 2$, and the threshold value is computed as $T_f = T_{f(x_8=0)} = 25$. Hence, we successfully obtain the LTG of function $f(x_1, \ldots, x_8)$ with the weight-threshold vector $[7, 6, 6, 4, 4, 4, 3, 2; 25]$. Since the weight-threshold vector of function f has been determined, f is identified as a TF.

4.4 New Initial Weight Assignment

In the last example in Section 4.3, we mentioned that the weight assignment algorithm can be the same as Reference [15] and the initial weight assignment assigned a least weight to each variable when searching a common weight vector of cofactor functions satisfying the VWO. However, we observe that the initial weight assignment can be set "larger" such that the weight assignment procedure for searching the common weight vector will be more efficient, thanks to the proposed Theorem 1. Hence, in this subsection, we present our method about the new initial weight assignment. The effect of this initial weight assignment can be seen in the experimental results.

When we search the common weight vector for the two cofactor functions, we may already have the original weight vector of each cofactor function. These two original weight vectors reveal that the weight assignments smaller than themselves have been checked as illegal weight assignments. Hence, we can consider to refer to these two original weight vectors for obtaining the initial weight assignment. Assume that the original weight vector of the n-input cofactor function $f(x_1, \ldots, x_n, x_{n+1} = 0)$ is $[w_{01}, w_{02}, \ldots, w_{0n}]$, and that of $f(x_1, \ldots, x_n, x_{n+1} = 1)$ is $[w_{11}, w_{12}, \ldots, w_{1n}]$. These two weight vectors might not be equal and they represent the lower

ALGORITHM 1: Initial Weight Assignment.

Require: Two weight vectors of the n-input cofactor functions $[w_{01}, w_{02}, \ldots, w_{0n}]$, $[w_{11}, w_{12}, \ldots, w_{1n}]$, and the VWO of the $(n+1)$-input composite function $[x_1, \ldots, x_n, x_{n+1}]$.
Ensure: The weight vector $[w_1, \ldots, w_n]$ that meets the weight lower bound.

```
for i = 1 ~ n do
    w_i ← max(w_0i, w_1i);
end for
for i = (n − 1) ~ 1 do
    if x_i > x_{i+1} & w_i ≤ w_{i+1} then
        w_i ← w_{i+1} + 1;
    else if x_i = x_{i+1} & w_i < w_{i+1} then
        w_i ← w_{i+1};
    end if
end for
for i = 1 ~ (n − 1) do
    if x_i = x_{i+1} then
        w_{i+1} ← w_i;
    end if
end for
```

bounds of legal weight assignments for each cofactor function. Thus, we propose to have the initial weight assignment $[w_1, w_2, \ldots, w_i, \ldots, w_n]$ that satisfies the following conditions:

(1) $w_i \geq w_{0i}$;
(2) $w_i \geq w_{1i}$;
(3) when $p_i(f) > p_j(f)$, $w_i > w_j$;
(4) when $p_i(f) = p_j(f)$, $w_i = w_j$;

where $p_i(f), p_j(f)$ are the elements in Chow's parameter of f, $1 \leq i, j \leq n$. Conditions (1) and (2) assume that the weights of the cofactor functions are the lower bounds of the corresponding weights in the composite function. Conditions (3) and (4) mean that the initial weight assignment must match the VWO of function f. The pseudo code of computing the initial weight assignment satisfying conditions (1) $\sim$ (4) is shown in Algorithm 1. Note that, since the VWO of first n input variables of the composite function $f(x_1, \ldots, x_n, x_{n+1})$ may not be identical to the VWO of each cofactor function, $f(x_1, \ldots, x_n, x_{n+1} = 0)$ or $f(x_1, \ldots, x_n, x_{n+1} = 1)$, the proposed Algorithm 1 is a heuristic for elevating the efficiency of weight assignment procedure. We use an example to explain the proposed initial weight assignment.

Given a 7-input function $f(x_1, \ldots, x_6, x_7)$ with the VWO $x_1 > x_2 > x_3 > x_4 > x_5 = x_6 > x_7$, its cofactor functions $f(x_1, \ldots, x_6, x_7 = 0)$ and $f(x_1, \ldots, x_6, x_7 = 1)$ are with the weight vectors $[9, 5, 4, 3, 2, 2]$ and $[8, 7, 7, 3, 2, 2]$, respectively. By comparing the two weight vectors, we have the initial weight assignment for the first 6 inputs as $[9, 7, 7, 3, 2, 2]$. To meet the VWO of f, we increase the 2nd weight from 7 to 8. As a result, the initial weight assignment is $[9, 8, 7, 3, 2, 2]$.

4.5 Overall Flow

The flowchart of the proposed TF identification algorithm is as shown in Figure 15. Given an $(n+1)$-input function $f(x_1, \ldots, x_n, x_{n+1})$ in ISOP form. We first check if the function f satisfies the necessary condition in Reference [13] or not. If the function f does not satisfy the necessary condition of being a TF, then it is identified as a non-TF. Then the algorithm generates the composite inequality system using the proposed approach presented in Section 4.2.

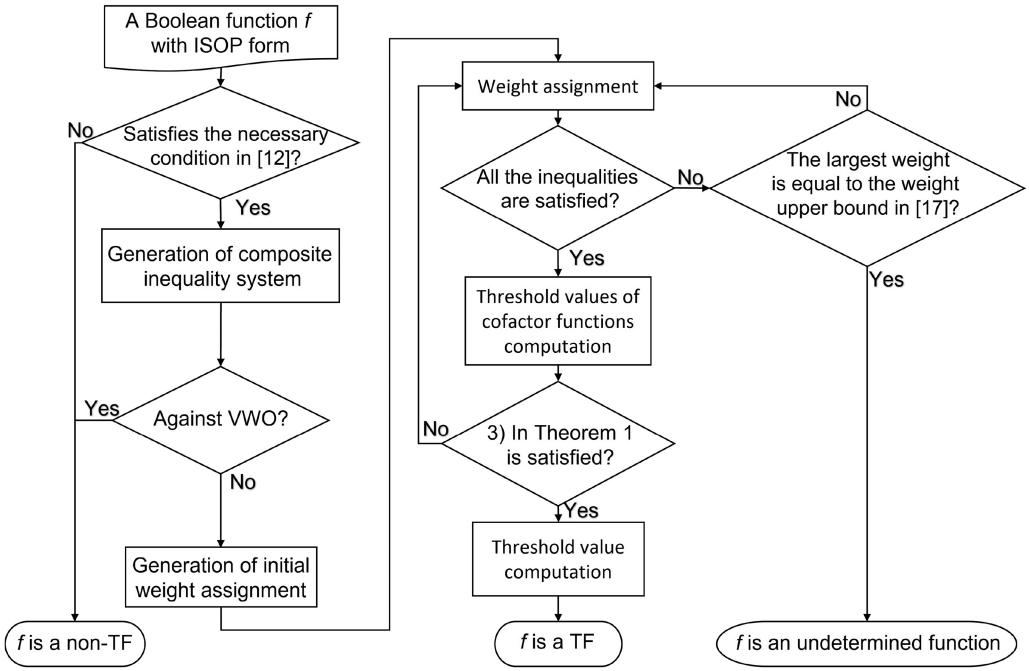

Fig. 15. The flowchart of the proposed algorithm for TF identification.

If there exists an inequality that violates the VWO, then the function f is identified as a non-TF. The algorithm assigns an initial weight assignment using the proposed method presented in Section 4.4. If a common weight vector satisfying the inequality system is found, then we compute the threshold values $T_{f(x_{n+1}=0)}$ and $T_{f(x_{n+1}=1)}$ of the cofactor functions $f(x_1, \ldots, x_n, x_{n+1} = 0)$ and $f(x_1, \ldots, x_n, x_{n+1} = 1)$ with the proposed method presented in Section 4.3. If the threshold values $T_{f(x_{n+1}=0)}$ and $T_{f(x_{n+1}=1)}$ satisfy (3) in Theorem 1, then the algorithm then computes the last weight w_{n+1} and the threshold value T_f of the function f, and f is identified as a TF; otherwise, the algorithm enters a weight adjustment loop until it finds another common weight vector. Note that in the weight assignment procedure, when an assigned weight is larger than the theoretically weight upper bound mentioned in Reference [17], the function is identified as an undetermined function.

5 TIME COMPLEXITY ANALYSIS

In this section, we analyze and compare the time complexity of weight assignment procedure in Reference [15] and our approach, which is the main part of TF identification algorithm. Given the number of on-set (off-set) minterms s (t) of a function, the theoretically upper bound of weight m, and a user-defined parameter X in each weight assignment process. When there are multiple weights w_i with $d(w_i) > 0$, we adjust the top X weights based on the magnitude of their d values. For example, if there are five weights with d values greater than zero and $X = 3$, then we only adjust three weights with the largest d values.

To perform the weight assignment procedure, we need to check whether $s * t$ inequalities hold for each assignment and adjust the assignment by one at a time. We repeat this process up to X^m times before reaching m. Therefore, the time complexity of this procedure is $O(s * t * X^m)$, which is the product of the number of inequalities and the number of attempted assignments. Hence, we know that the CPU time is highly related to the number of inequalities in the inequality system.

Table 1. Result Comparison Using Different Initial Weight Assignments

|Input|	|T. TF|	Initiate with [15]	Initiate with [15] and Ours		Initiate with Ours		
		CPU(s)	CPU(s)	Overhead(s)	|Id. TF|	CPU(s)	Overhead(s)
6	994	0.01	0.03	0.02	994	0.04	0.02
7	28,262	4.27	5.24	1.60	28,262	4.48	1.96
8	2,700,791	**56,985**	33,138	902	2,700,791	13,469	780
Total	–	**56,989**	33,143	–	–	13,474	–
Ratio	–	1	0.58	–	–	0.23	–

In Reference [15], the number of inequalities in the inequality system for an $(n+1)$-input function is $s*t$. Since we know $\frac{s+t}{2} \geq \sqrt{s*t}$, the maximal value of $s*t$ is $(\frac{s+t}{2})^2$. Since s and t represent the numbers of on-set and off-set minterms of an $(n+1)$-input function, $s+t$ contains all 2^{n+1} minterms. The upper bound of the number of inequalities in the inequality system is $(\frac{2^{n+1}}{2})^2 = (2^n)^2 = 2^{2n}$. In our approach, the inequality system is built from two n-input cofactor functions. As a result, the number of inequalities in our inequality system is smaller than or equal to $(\frac{2^n}{2})^2 * 2$, which is 2^{2n-1}.

As compared to Reference [15], our weight assignment procedure only constructs a half number of inequalities theoretically in the inequality system, which results in performance improvement as shown in the experimental results.

6 EXPERIMENTAL RESULTS

We implemented the proposed algorithm in C++ language. Except for the fourth experiment, the experiments were mainly conducted on a 2.9 GHz Linux platform (CentOS 7.9).

We conducted four experiments in this work. The first one is to show the efficiency and the effectiveness of the proposed new initial weight assignment. The second experiment is to show the efficiency of our TF identification algorithm and the optimality of computed weight-threshold vector of the identified TFs. The optimal LTG in this work is defined as the LTG with the optimal weight-threshold vector that is obtained by ILP-based approach [26]. The third one shows that the proposed approach is also effective and efficient for identifying TFs with more than eight inputs. The last one shows that our approach demonstrated significantly faster performance than the ILP-based method for common threshold logic networks with 6-input TFs.

For the first experiment, we conducted the proposed TF identification algorithm using different initial weight assignments for functions with 6 to 8 inputs. For the functions with fewer inputs, the difference is unobvious. For fairness, we assume that our approach does not have n-input TFs at hand when identifying $(n+1)$-input TFs. Hence, when computing the initial weight assignment for an $(n+1)$-input function $f(x_1, \ldots, x_n, x_{n+1})$, we need to compute the weight vectors of two n-input cofactor functions $f(x_1, \ldots, x_n, x_{n+1} = 0)$ and $f(x_1, \ldots, x_n, x_{n+1} = 1)$ recursively. We call this extra cost the CPU time overhead. Table 1 shows the experimental results, including the required CPU time and the CPU time overhead. Column 1 is the number of inputs. Column 2 is the total number of TFs. Column 3 shows the required CPU time for identifying all these TFs using the initial weight assignment from Reference [15], which is the least weight vector. Column 4 shows the required CPU time when the weight assignment procedure starts with both the least weight vector from Reference [15] and our initial weight vector mentioned in Section 4.4. An example procedure is shown in Figure 16. The left part of Figure 16 is our initial weight vector $(6, 4, 2, 1)$, while the right part is the least weight vector $(4, 3, 2, 1)$. The checking sequence of weight vectors is summarized on the right of Figure 16, which selects the weight vector from both parts alternatively. Column 7 shows the required CPU time only using our initial weight vector. Columns 5 and 8 list the CPU time overhead for computing the weight vectors of the cofactor functions, which is included in the CPU time. Column 6 shows the number of TFs identified with our initial weight assignment.

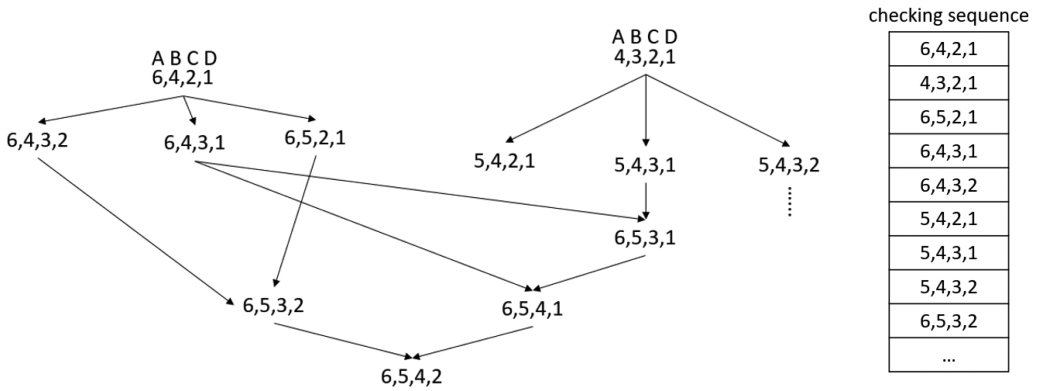

Fig. 16. An example procedure of weight assignments with our initial weight vector and the least weight vector from [15].

Table 2. CPU Time and the Number of Obtained LTGs Comparison between [15] and Our Approach

|Input|	|T. TF|	[15]				Ours				
		|Id. TF|	CPU(s)	|Opt. LTG|	Ratio	|Id. TF|	CPU(s)	Ratio	|Opt. LTG|	Ratio
4	17	17	<0.01	17	1	17	<0.01	–	17	1
5	92	92	<0.01	92	1	92	<0.01	–	92	1
6	994	994	0.01	994	1	994	0.05	5	994	1
7	28,262	28,262	5.19	28,249	0.9995	28,262	4.48	0.86	28,254	0.9997
8	2,700,791	2,700,791	68,616	2,695,746	0.9981	2,700,791	13,469	0.20	2,698,521	0.9991

According to Table 1, when using the initial weight assignment proposed in Reference [15], we spent 56, 985 seconds to identify all the 8-input TFs. Similarly, we spent 33, 138 seconds to identify all the 8-input TFs with the proposed initial weight assignment and that proposed in Reference [15]. When using the proposed initial weight assignment, only 13, 469 seconds are required to identify all the 8-input TFs. When considering all the benchmarks, our initial weight assignment saves 77% CPU time. This result indicates that the proposed heuristic of initial weight assignment is efficient and effective for identifying all the NP-class with up to 8 inputs. However, we also notice that our initial weight assignment spent more CPU time than Reference [15] for 6-input TFs. The reason is that the weights in 5-input TFs are very small such that our initial weight assignment is only a bit larger than that in Reference [15]. As a result, our approach incurred CPU time overhead for computing the weight vectors of the cofactor functions.

The proposed TF identification algorithm has two features: one is the more simplified composite inequality system based on Theorem 1; the other is the new initial weight assignment. Thus, in the second experiment, we want to demonstrate the effectiveness and efficiency improvement of our approach with the proposed features. That is, we compare the CPU time and the optimality of the obtained LTGs in our approach against Reference [15] for identifying all the NP-class TFs with 4 to 8 inputs. The experimental results are shown in Table 2. According to Table 2, both our approach and the state-of-the-art [15] identified all the NP-class TFs within 8 inputs. However, our approach only spent 13, 469 seconds to identify all the 8-input TFs while Reference [15] spent 68, 616 seconds. The CPU time reduction of our approach is 80%. We also observe that 2,700 more optimal LTGs are obtained by our approach as compared to the state-of-the-art [15]. Table 3 shows the comparison of the number of inequalities between the state-of-the-art [15] and ours. The number of inequalities in our approach is only about 66% of that in the state-of-the-art [15]. This is the root cause of CPU time-saving in our approach. In summary, our approach is more efficient and obtains better results in identifying TFs than the state-of-the-art [15].

Table 3. Number of Inequality Comparison between [15] and Our Approach

|Input|	[15]	Ours	Ratio
4	6	4	0.66
5	140	84	0.60
6	4,808	3,014	0.62
7	395,744	258,010	0.65
8	104,702,204	66,425,928	0.63

Table 4. CPU Time Comparison between [15] and Our Approach with 9–15 Inputs

|Input|	[15] CPU(s)	Ours CPU(s)	Ratio
9	407.64	165.21	0.41
10	2,175.06	502.67	0.23
11	5,711.39	1,279.99	0.22
12	11,401.10	3,398.37	0.30
13	20,717.80	9,293.14	0.45
14	28,070.90	26,731.00	0.95
15	45,405.50	85,614.46	1.89

For the third experiment, we conducted our approach for TFs with 9 to 15 inputs. Since most of the synthesized threshold circuits do not contain LTGs with more than 15 inputs, we identify TFs with 9 to 15 inputs to demonstrate the applicability of our approach. The total number of TFs from 9 to 15 inputs is extremely enormous. Therefore, we randomly generate 100, 000 TFs for each category. The results are summarized in Table 4. Since all the generated TFs can be identified by the both approaches, we only report the required CPU time. According to Table 4, our approach can identify the TFs with 9 to 14 inputs more efficiently. However, to identify the TFs with 15 inputs, our approach needs more CPU time. The reason is that our approach recursively identifies $f(x_1, \ldots, x_n, x_{n+1} = 0)$ and $f(x_1, \ldots, x_n, x_{n+1} = 1)$ when identifying $f(x_1, \ldots, x_n, x_{n+1})$. Hence, as n increases one, the required CPU time will be roughly double to triple. That is, the CPU time of our approach increases exponentially with respect to the growth of n. Thus, the proposed approach is more appropriate to identifying TFs with less than 15 inputs.

For the last experiment, we compare the performance of using ILP solvers to find shared weights. Due to version compatibility issues with the ILP solvers, we conducted our comparison on a 2.8 GHz Linux platform (CentOS 7.9) using Gurobi 9.5.2 [1]. We randomly generated 10, 000 test cases for TFs with 9 to 15 inputs. The experimental results are shown in Table 5. According to Table 5, our approach was better than the ILP-based method in terms of CPU time for TFs with 9 or fewer inputs. For common threshold logic networks with 6-input TFs, our approach demonstrated significantly faster performance than the ILP-based method. However, for TFs with 10 to 15 inputs, our approach becomes slower due to the algorithm's need for recursion. At present, we have not yet found a TF library for 9 inputs available. Nevertheless, our approach shows promise in helping build such a library in the future.

7 CONCLUSION AND FUTURE WORK

In this article, we propose a new sufficient and necessary condition for a function being a TF and devise a TF identification algorithm with this condition. We also present a new initial weight assignment method to accelerate the weight assignment procedure. The experimental results show

Table 5. CPU Time Comparison between [1] and Our Approach with 4–15 Inputs

|Input|	|T. TF|	[1] CPU(s)	Ours CPU(s)	Ratio
4	17	0.09	<0.01	0.11
5	92	0.49	<0.01	0.02
6	994	5.53	0.03	0.01
7	28,262	173.79	4.64	0.03
8	2,700,791	17,326.19	10,286.40	0.59
9	10,000	80.05	54.63	0.68
10	10,000	113.77	171.19	1.50
11	10,000	177.09	469.30	2.65
12	10,000	380.89	1,229.39	3.23
13	10,000	987.44	3,291.15	3.33
14	10,000	3,349.81	10,158.20	3.03
15	10,000	12,005.50	31,809.60	2.65

that the proposed approach saves 80% CPU time and obtains more optimal LTGs with smaller weights and threshold values for identifying all the NP-class TFs with 8 inputs. In addition to the acceleration of identifying TFs, the proposed sufficient and necessary condition constructs all $(n+1)$-input TFs from their n-input cofactor TFs. Our future work is to identify all the 9-input NP-class TFs based on the proposed sufficient and necessary condition, which will be the first attempt in TF identification.

REFERENCES

[1] Gurobi. https://www.gurobi.com/.

[2] M. J. Avedillo, J. M. Quintana, H. Pettenghi, P. M. Kelly, and C. J. Thompson. 2003. Multi-threshold threshold logic circuit design using resonant tunnelling devices. *Electron. Lett.* 39, 21 (2003), 1.

[3] Yung-Chih Chen, Hao-Ju Chang, and Li-Cheng Zheng. 2020. Don't-care-based node minimization for threshold logic networks. In *57th ACM/IEEE Design Automation Conference (DAC'20)*. IEEE, 1–6.

[4] Calvin C. Elgot. 1961. Truth functions realizable by single threshold organs. In *2nd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT'61)*. IEEE, 225–245.

[5] Sukumar Ghosh, Subir Bandyopadhyay, Sanjit Kumar Mitra, and Arun Kumar Choudhury. 1972. Simple methods for the testing of 2-summability of Boolean functions and isobaricity of threshold functions. *IEEE Trans. Comput.* 100, 5 (1972), 503–507.

[6] David Goldhaber-Gordon, Michael S. Montemerlo, J. Christopher Love, Gregory J. Opiteck, and James C. Ellenbogen. 1997. Overview of nanoelectronic devices. *Proc. IEEE* 85, 4 (1997), 521–540.

[7] Wen-Chih Hsu, Chia-Chun Lin, Yi-Ting Li, Yung-Chih Chen, and Chun-Yao Wang. 2021. On reduction of computations for threshold function identification. In *IEEE 34th International System-on-Chip Conference (SOCC'21)*. IEEE, 146–151.

[8] Pin-Yi Kuo, Chun-Yao Wang, and Ching-Yi Huang. 2011. On rewiring and simplification for canonicity in threshold logic circuits. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'11)*. IEEE, 396–403.

[9] Siang-Yun Lee, Nian-Ze Lee, and Jie-Hong R. Jiang. 2018. Canonicalization of threshold logic representation and its applications. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'18)*. IEEE, 1–8.

[10] Siang-Yun Lee, Nian-Ze Lee, and Jie-Hong R. Jiang. 2019. Searching parallel separating hyperplanes for effective compression of threshold logic networks. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'19)*. IEEE, 1–8.

[11] Chia-Chun Lin, Chiao-Wei Huang, Chun-Yao Wang, and Yung-Chih Chen. 2017. In&Out: Restructuring for threshold logic network optimization. In *18th International Symposium on Quality Electronic Design (ISQED'17)*. IEEE, 413–418.

[12] Chia-Chun Lin, Ciao-Syun Lin, You-Hsuen Tsai, Yung-Chih Chen, and Chun-Yao Wang. 2021. Don't care computation and De Morgan transformation for threshold logic network optimization. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 41, 5 (2021), 1412–1422.

[13] Chia-Chun Lin, Chin-Heng Liu, Yung-Chih Chen, and Chun-Yao Wang. 2020. A new necessary condition for threshold function identification. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 39, 12 (2020), 5304–5308.
[14] Chia-Chun Lin, Chun-Yao Wang, Yung-Chih Chen, and Ching-Yi Huang. 2014. Rewiring for threshold logic circuit minimization. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'14)*. IEEE, 1–6.
[15] Chin-Heng Liu, Chia-Chun Lin, Yung-Chih Chen, Chia-Cheng Wu, Chun-Yao Wang, and Shigeru Yamashita. 2018. Threshold function identification by redundancy removal and comprehensive weight assignments. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 38, 12 (2018), 2284–2297.
[16] Vassilios A. Mardiris, Georgios Ch. Sirakoulis, and Ioannis G. Karafyllidis. 2014. Automated design architecture for 1-D cellular automata using quantum cellular automata. *IEEE Trans. Comput.* 64, 9 (2014), 2476–2489.
[17] Saburo Muroga. 1971. *Threshold Logic and Its Applications*. Wiley, New York, NY.
[18] Saburo Muroga, I. Toda, and M. Kondo. 1962. Majority decision functions of up to six variables. *Math. Comput.* 16, 80 (1962), 459–472.
[19] Saburo Muroga, Teiichi Tsuboi, and Charles Richmond Baugh. 1970. Enumeration of threshold functions of eight variables. *IEEE Trans. Comput.* 100, 9 (1970), 818–825.
[20] Augusto Neutzling, Mayler G. A. Martins, Vinicius Callegaro, Andre I. Reis, and Renato P. Ribas. 2017. A simple and effective heuristic method for threshold logic identification. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 37, 5 (2017), 1023–1036.
[21] Georgios Papandroulidakis, Alexander Serb, Ali Khiat, Geoff V. Merrett, and Themis Prodromakis. 2019. Practical implementation of memristor-based threshold logic gates. *IEEE Trans. Circ. Syst. I: Reg. Pap.* 66, 8 (2019), 3041–3051.
[22] Vinay Saripalli, Lu Liu, Suman Datta, and Vijaykrishnan Narayanan. 2010. Energy-delay performance of nanoscale transistors exhibiting single electron behavior and associated logic circuits. *J. Low Power Electron.* 6, 3 (2010), 415–428.
[23] Anil K. Sarje and Nripendra N. Biswas. 1979. A new approach to 2-asummability testing. *IEEE Trans. Comput.* 28, 10 (1979), 798–801.
[24] Chen-Kuan Tsai, Chun-Yao Wang, Ching-Yi Huang, and Yung-Chih Chen. 2013. Sensitization criterion for threshold logic circuits and its application. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'13)*. IEEE, 226–233.
[25] Robert O. Winder. 1961. Single stage threshold logic. In *2nd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT'61)*. IEEE, 321–332.
[26] Rui Zhang, Pallav Gupta, Lin Zhong, and Niraj K. Jha. 2004. Threshold network synthesis and optimization and its application to nanotechnologies. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 24, 1 (2004), 107–118.

Received 29 November 2022; revised 24 May 2023; accepted 8 June 2023