On Reduction of Computations for Threshold Function Identification

Wen-Chih Hsu, Chia-Chun Lin, Yi-Ting Li, Yung-Chih Chen, and Chun-Yao Wang, *Member, IEEE*

2021 IEEE 34th International System-on-Chip Conference (SOCC) | 978-1-6654-2931-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/SOCC52499.2021.9739224

***Abstract*—Knowing a sufficient and necessary condition for being a threshold function (TF) is quite crucial for TF identification algorithm. However, to the best of our knowledge, no one proposed an efficient sufficient and necessary condition for being a TF. Hence, the state-of-the-art approach to this identification problem exploits a necessary condition, and a weight and threshold value assignment approach to identify TFs instead. In fact, a sufficient and necessary condition for being a TF had been proposed many decades ago, which is called the Summable Theorem. However, this theorem and the corresponding checking algorithm are not practical from the viewpoint of efficiency due to the high complexity. As a result, in this work, we propose several theorems such that the complexity of the TF identification algorithm can be significantly reduced. Furthermore, according to the experimental results, 76%∼96% computations are saved on average, and 40%∼75% CPU time are saved on average, for a set of 6-input∼9-input unate functions.**

I. Introduction

Threshold logic has been attracting great attention from researchers due to its similarity to an artificial neuron. As comparing with Boolean logic, threshold logic can concisely represent the thresholding behavior of an artificial neuron in the Neural Networks (NNs) [1]. Additionally, the rapid development of emerging technologies such as Resonant Tunneling Diodes [2], Quantum Celluar Automata [11], Single Electron Transistors [20], and Memristors [17] also resulted in the flourishing research of threshold logic.

A Linear Threshold Gate (LTG) is the primitive element in the threshold logic. The function f of an LTG with n binary inputs, x_1,...,x_n, is defined as EQ(1).

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

Each binary input x_i has a corresponding weight w_i. If $\sum_{i=1}^{n} x_i w_i$ is greater than or equal to the threshold value T, the output of f is 1; otherwise, the output of f is 0. A function that can be represented by an LTG is called a *Threshold Function (TF)*. For example, a function $f = x_1x_2 + x_1x_3$ is a TF and the corresponding LTG is shown in Fig. 1(a). However, a function $g = x_1x_2 + x_3x_4$ is not a TF, which is called a *non-TF*. Fig. 1(b) shows the corresponding threshold network that consists of more than one LTG to represent the non-TF g. A fundamental but important topic in threshold logic is to determine whether a given function is a TF or not. This is because many applications, such as threshold logic synthesis [15] and equivalence checking [6], need efficient and effective algorithms for TF identification.

Recently, several heuristics for TF identification have been proposed. For example, the authors of [16] proposed a method that first generated an inequality system about the weights and threshold value from a given function. Then by simplifying this inequality system and incrementally assigning the weights and threshold value in the simplified inequality system, the weights and threshold value of an LTG might be obtained. Once the weights and threshold value satisfying EQ(1) are found, the function is a TF and the corresponding LTG can be realized.

Later, the authors of [10] observed the *flip situation* in the TF identification algorithm of [16]. The TF identification algorithm with the *flip situation* might fail to identify some given TFs. Therefore, a more comprehensive method was proposed in [10], which provided a new weight assignment procedure and improved the results of TF identification algorithm.

Traditionally, *unateness* is a necessary condition of a TF. That is, all the TFs are *unate*. If a function is not unate, it is a non-TF. Recently, a theoretic study [9] proposed a new necessary condition of a TF, which is more precise than the unateness such that more non-TFs can be removed in the early stage of the TF identification algorithm [10].

However, there still may exist some non-TFs that cannot be pruned out by this new necessary condition. Those non-TFs still enter the stage of weight assignment though. Since the weight and threshold value cannot be obtained intrinsically for a non-TF, the TF identification algorithm will return *undetermined* when the algorithm cannot report a conflict in the inequality system [10]. Thus, knowing a sufficient and necessary condition for being a TF is quite crucial for TF identification algorithm. In fact, many decades ago, the author of [13] had proposed a sufficient and necessary condition for being a TF, which is called the *Summable Theorem*. However, this theorem and the corresponding checking algorithm are not practical from the viewpoint of efficiency due to high complexity. Therefore, in this paper, we propose several theorems such that the complexity of the TF identification algorithm can be significantly reduced.

The main contributions of this work are twofold:

1) We propose an improved Summable Theorem, which is called the *Semi-Critical Summable Theorem*, for unate func-

This work is supported in part by the Ministry of Science and Technology of Taiwan under MOST 106-2221-E-007-111-MY3, MOST 109-2221-E-007-082-MY2, MOST 109-2221-E-155-047-MY2, and MOST 109-2224-E-007-005.

W.-C. Hsu, C.-C. Lin, Y.-T. Li and C.-Y. Wang are with the Department of Computer Science, National Tsing Hua University, Hsinchu 300044, Taiwan R.O.C. Y.-C. Chen is with the Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei 106335, Taiwan R.O.C.

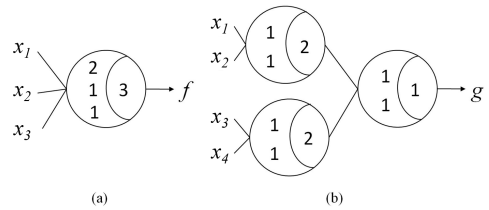

Fig. 1. (a) A Boolean function $f = x_1x_2 + x_1x_3$ represented by an LTG. (b) The threshold network of function $g = x_1x_2 + x_3x_4$.

tions. This research advances the theoretic study about threshold functions.

2) In comparison to the Summable Theorem, 76%∼96% of computations can be reduced on average and 40%∼75% of CPU time can be reduced for unate functions with 6 to 9 inputs.

II. Preliminaries

In this section, we review the background of this work.

A. Unate Function

Unate function is a subset of Boolean function with a monotonicity property. An n-input function $f(x_1, x_2, ..., x_i, ..., x_n)$ is said to be *positive unate in variable* x_i if and only if for all possible values in other variable x_j, where $j \neq i$,

$$f(x_1, x_2, ..., x_{i-1}, x_i = 1, x_{i+1}, ..., x_n) \geq f(x_1, x_2, ..., x_{i-1}, x_i = 0, x_{i+1}, ..., x_n). \quad (2)$$

Similarly, an n-input function $f(x_1, x_2, ..., x_i, ..., x_n)$ is said to be *negative unate in variable* x_i if and only if

$$f(x_1, x_2, ..., x_{i-1}, x_i = 0, x_{i+1}, ..., x_n) \geq f(x_1, x_2, ..., x_{i-1}, x_i = 1, x_{i+1}, ..., x_n). \quad (3)$$

If f is either positive or negative unate in all the variables, f is said to be an *unate function* or the function has the unateness property. For example, $x_1x_2 + x_1x_3$ and $x_1'x_2 + x_1'x_3$ are both unate functions, while $x_1'x_2 + x_1x_3$ is not a unate function. If f is *positive (negative) unate* in all the variables, it is said to be a *positive (negative) unate* function. Since we can obtain an equivalent NPN-class function by simply permuting, or negating the inputs or outputs of the given function [13], we just consider positive unate functions in this work.

B. Threshold Function

TF is a subset of Boolean function that can be represented by an LTG. All the TFs are unate functions. Since unateness is a necessary condition for being a TF, we can exploit unateness to accelerate the TF identification algorithm. That is, if a function is not unate, it is a non-TF.

C. Summable Theorem

Summable Theorem was proposed in [13], which can be used to identify whether a given function is a TF or not by checking the equality of summation of some input vectors. Let f^{on} denote the *on-set* of f and f^{off} denote the *off-set* of f.

Definition 1 [13]**:** A function f is said to be k-summable if and only if for some k, $2 \leq k$, there exist two sets $A \subseteq f^{on}$, $B \subseteq f^{off}$ such that $\{A^{(j)}|f(A^{(j)}) = 1\}$, $\{B^{(j)}|f(B^{(j)}) = 0\}$, and

$$\sum_{j=1}^{k} A^{(j)} = \sum_{j=1}^{k} B^{(j)} \quad (4)$$

provided that repetition of vectors in the sets A and B are permitted. If f is k-summable for some k, f is said to be *summable* for short; otherwise, f is said to be *asummable*.

Theorem 1 (Summable Theorem) [13]**:** A function f is a TF if and only if f is asummable.

Proof [13]**:** Omitted. ■

Corollary 1: A function f is a non-TF if and only if f is summable.

Proof: By the contraposition of Theorem 1, we have Corollary 1. ■

Theorem 1 states that asummability is a sufficient and necessary condition for being a TF. For example, given a Boolean function $f = x_1x_2+x_1x_3$. Since there does not exist $2 \leq k$ such that $\sum_{j=1}^{k} A^{(j)} = \sum_{j=1}^{k} B^{(j)}$, where $A \subseteq f^{on}$ and $B \subseteq f^{off}$, f is asummable. Thus, f is a TF. On the other hand, given a Boolean function $g = x_1x_2 + x_3x_4$. When considering two vectors $A^{(j)}$ from g^{on}, i.e., $A^{(1)} = 1100, A^{(2)} = 0011$, and two vectors $B^{(j)}$ from g^{off}, i.e., $B^{(1)} = 1010, B^{(2)} = 0101$, we have $\sum_{j=1}^{2} A^{(j)} = 1111 = \sum_{j=1}^{2} B^{(j)}$. Thus, g is 2-summable, i.e., summable by Definition 1. According to Corollary 1, g is a non-TF.

Although Theorem 1 reveals the sufficient condition for being a TF, we barely use it to identify whether a function is a TF or not in practice. This is because this checking procedure will never be terminated. For example, given a Boolean function $f = x_1x_2 + x_1x_3$. If we want to know that f is a TF by Theorem 1, we need to know that f is asummable. That is, we have to confirm that there does not exist sets A and B satisfying EQ(4) for all $2 \leq k$. Furthermore, since the repetition of vectors in the sets A and B are permitted, no upper bound exists for k. As a result, this checking procedure will never be terminated. This is also the reason why the previous TF identification algorithms [10][16] exploited weight and threshold value assignment procedure to identify a TF. That is, when the weights and threshold value of an LTG with respect to a function are found, the function is a TF.

D. A New Necessary Condition for Threshold Function

Recently, the work [9] proposed another theoretical study about the necessary condition for being a TF.

Definition 2 [9]**:** Two functions f and g have the implication relation if and only if $f \subseteq g$ or $g \subseteq f$.

Theorem 2 [9]**:** If a function f is a TF, then its two cofactor functions, $f(x_i = 1, x_j = 0)$ and $f(x_i = 0, x_j = 1)$ have the implication relation, for all input pairs x_i, x_j.

Proof [9]**:** Omitted. ■

From the contraposition of Theorem 2, we can know that for a function f, if there exist two cofactor functions, $f(x_i = 1, x_j = 0)$ and $f(x_i = 0, x_j = 1)$, that do not have the implication relation for an input pair x_i and x_j, then f is a non-TF. For example, given a function $f = ab + ad + bc$. We have the following two cofactor functions, $f(a = 1, b = 0) = d$ and $f(a = 0, b = 1) = c$. Since $d \nsubseteq c$ and $c \nsubseteq d$, the two cofactor functions do not have the implication relation. Thus, the function $f = ab + ad + bc$ is a non-TF.

As compared to unateness, the implication relation in Theorem 2 is more precise. Many unate functions that are non-TFs, as the last example, can be successfully identified by Theorem 2.

III. Semi-critical Summable Theorem

In this section, we present the proposed Semi-Critical Summable Theorem that can significantly reduce the time complexity of the TF identification exploiting *Summable Theorem* mentioned in Theorem 1.

A. Time Complexity of Summable Theorem

To determine if an n-input function f, which contains s vectors $\in f^{on}$ and t vectors $\in f^{off}$ and $s + t = 2^n$, is summable or not, we have to compute all the combinations in EQ(4). That is, we first check whether the function is 2-summable by selecting two vectors from f^{on} and two vectors from f^{off}. Since these two vectors can be chosen repeatedly according to Definition 1, the time complexity of checking 2-summable for the function f is $O(s^2t^2)$. Similarly, the time complexity of checking k-summable for the function f is $O(s^kt^k)$. Obviously, this summable checking procedure is computation-intensive, especially when the sizes of f^{on} and f^{off} are getting larger. Furthermore, there does not exist an upper bound about k for this checking procedure because the vector repetition in the sets A and B are permitted.

Since the unateness property is a necessary condition for being a TF, we can easily identify the given functions as non-TFs if they are not unate. Thus, to demonstrate the strength of the proposed theorem, we assume that all the functions to be identified are positive unate functions. In the next subsection, we will introduce how to reduce the sizes of sets A and B in EQ(4) for improving the efficiency of the summable checking procedure.

B. Set Size Reduction in the Summable Theorem

Before introducing how to reduce the set sizes in the *Summable Theorem*, we first introduce the definitions of the *ON Critical Vectors* (*ONCVs*) and *OFF Critical Vectors* (*OFFCVs*) of a function.

Definition 3: Given an input vector $v \in f^{on}$ (f^{off}) of a positive unate function f, v is said to be an ONCV (OFFCV) if and only if every single bit of v flips from *1 to 0* (*0 to 1*), the output of f also flips from *1 to 0* (*0 to 1*).

For example, Fig. 2(a) shows the truth table of a positive unate function $f = a + bc$. According to Definition 3, we know that the input vectors 011 and 100 are ONCVs, and the input vectors 001 and 010 are OFFCVs.

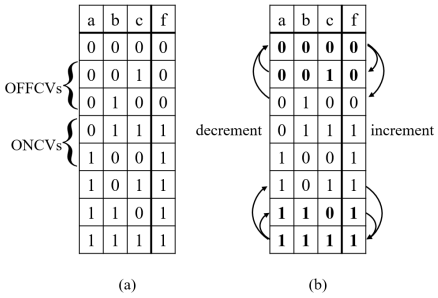

Fig. 2. (a) The truth table of $f = a + bc$ and the ONCVs and OFFCVs of f. (b) The input vector 110 can be obtained by applying a decrement operation on the input vector 111. The input vector 001 can be obtained by applying an increment operation on the input vector 000.

Next, we define two operations, *decrement* and *increment*, for an input vector v that turn v into different vectors by flipping one of its input bits.

Definition 4: Given an input vector v in a positive unate function f, the decrement (increment) operation is defined as flipping one bit in v from 1 to 0 (0 to 1) to obtain another input vector $\hat{v}$, and $f(v) = f(\hat{v})$.

Fig. 2(b) shows some decrement and increment operations. For example, the input vector 110 can be obtained from the input vector 111 by applying a decrement operation. Note that both vectors 110 and 111 have the same output value. Similarly, the input vector 001 can be obtained from the input vector 000 by applying an increment operation.

Lemma 1: Given a positive unate function f, for each vector $v \in f^{off}$, except for the all-0 vector, we can apply a decrement operation on it.

Proof: Applying a decrement operation on an input vector means that the output will be intact after the operation. Since the function f is positive unate, and according to its definition in EQ(2), assume that we have

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

for any x_i. Furthermore, a non-all-0 vector $v \in f^{off}$ always has the output of 0. Hence, without loss of generality, we can have

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

When we flip the input bit x_i from 1 to 0, we will have

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

based on EQ(2). Thus, we can apply a decrement operation on v. ■

Lemma 2: Given a positive unate function f, for each vector $v \in f^{on}$, except for the all-1 vector, we can apply an increment operation on it.

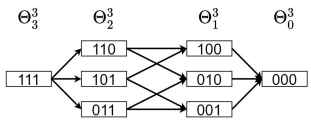

Fig. 3. The relationship among all the input vectors of a 3-input function.

Proof: Lemma 2 can be proved in a similar way as Lemma 1. Thus, this proof is omitted. ■

Theorem 3: Given a non-constant positive unate function f. An ONCV of f can be obtained by applying finite times of decrement operations on a non-ONCV $\in f^{on}$.

Proof: Let Θ_i^n denote a set of n-input vectors where the number of 1 in each input vector is i. For example, Θ_1^4 contains four input vectors 0001, 0010, 0100, and 1000. We can construct a directed acyclic graph G to represent the relationship among all the input vectors. An edge (u, v) in G indicates that an input vector u flips one of its input bits from 1 to 0 to turn into v. For example, Fig. 3 illustrates the relationship among all the input vectors of a 3-input function.

According to the construction rule in G, a vector $u \in \Theta_i^n$ has i edges connecting to i different vectors in Θ_{i-1}^n, which are denoted as v_j, $0 < j \leq i$. If $f(u) = 1$ and $f(v_j) = 0$ for all $0 < j \leq i$, then u is an ONCV by Definition 3. For example, each vector in Θ_2^3 has two edges connecting to the vectors in Θ_1^3. If $f(110) = 1$ and $f(100) = 0, f(010) = 0$, the input vector 110 is an ONCV. Note that Θ_n^n and Θ_0^n are the input vectors consisting of n 1s and 0s, respectively. For example, $\Theta_3^3 = 111$ and $\Theta_0^3 = 000$.

Now, we are going to prove that an ONCV can be obtained by applying finite times of decrement operations on a non-ONCV $\in f^{on}$. That is, we will reach to an ONCV by traversing G along the edges from a node representing a non-ONCV $\in f^{on}$. Traversing an edge (u, v) where $f(u) = f(v)$ means applying a decrement operation on u.

Since f is a positive unate function, and by the construction rule of graph G, we know that there does not exist an edge (u, v) with $f(u) = 0$ and $f(v) = 1$. That is, the edges in the graph G only cause three types of changes in the output, namely, $1 \to 1$, $1 \to 0$, and $0 \to 0$. Since f is a non-constant positive unate function, we have $f(\Theta_n^n) = 1$ and $f(\Theta_0^n) = 0$. This is because if $f(\Theta_n^n) = 0$, then f is a constant 0 function; and if $f(\Theta_0^n) = 1$, then f is a constant 1 function. Given a non-ONCV $\in f^{on}$, it means that there exists an input bit of the non-ONCV such that the bit flips from 1 to 0 but the output is still 1. Therefore, we can conduct a decrement operation on the non-ONCV.

By contradiction, assume that we cannot obtain an ONCV by applying finite times of decrement operations on a non-ONCV $\in f^{on}$. When the non-ONCV is in Θ_n^n ($f(\Theta_n^n) = 1$), we can apply a decrement operation on it, and turn it into a vector $p \in \Theta_{n-1}^n$ ($f(p) = 1$). Since p is still a non-ONCV by the hypothesis, we can apply a decrement operation on it again and turn it into a vector $p' \in \Theta_{n-2}^n$ ($f(p') = 1$), and so on. Finally, we will reach to a vector $u \in \Theta_1^n$ ($f(u) = 1$). Since u is still a non-ONCV, it means that there exists one edge (u, v) such that $v \in \Theta_0^n$ and $f(u) = 1$, $\mathbf{f}(\mathbf{v}) = \mathbf{1}$. However, we have known that there is only one vector $v \in \Theta_0^n$ and $\mathbf{f}(\mathbf{v}) = \mathbf{0}$, which contradicts $f(v) = 1$.

Thus, for a non-ONCV $\in f^{on}$, it will be turned into an ONCV by applying finite times of decrement operations. ■

Theorem 4: Given a non-constant positive unate function f. An OFFCV of f can be obtained by applying finite times of increment operations on a non-OFFCV $\in f^{off}$.

Proof: Theorem 4 can be proved in a similar way as Theorem 3. Thus, this proof is omitted. ■

By Theorem 3 and Lemma 1, we know that we can apply decrement operations on any non-ONCVs $\in f^{on}$ and any non-all-0 vectors $\in f^{off}$. According to *Summable Theorem* as mentioned in Definition 1 and Corollary 1, there exists one pair of vector sets $A \subseteq f^{on}$ and $B \subseteq f^{off}$ such that $\sum_{j=1}^{k} A^{(j)} = \sum_{j=1}^{k} B^{(j)}$ for a non-TF. Thus, we can apply one decrement operation on a non-ONCV in A and one decrement operation on a non-all-0 vector in B. When the decrement operation is applied on the same bit for both sets, the summation of vectors in A and the summation of vectors in B will be still equal. As a result, we can apply finite times of decrement operations on all the non-ONCVs in the set A and the corresponding vectors in the set B until all the non-ONCVs in A are turned into ONCVs. That is, if a function f is summable, there exist two sets A, B such that all the vectors in the set A are *only* ONCVs, all the vectors in the set $B \subseteq f^{off}$, and the summations of vectors in the set A, in the set B are equal. With this derivation, the set A can be reduced to containing ONCVs only.

Similarly, by Theorem 4 and Lemma 2, we can apply one increment operation on a non-all-1 vector $\in f^{on}$ in A and one increment operation on a non-OFFCV $\in f^{off}$ in B such that the summations of vectors in the sets A and B are still equal. After finite times of increment operations on both sets, all the non-OFFCVs in the set B are turned into OFFCVs. That is, if a function f is summable, then there exist two sets A, B such that all the vectors in the set $A \subseteq f^{on}$, all the vectors in the set B are *only* OFFCVs, and the summations of vectors in the set A, in the set B are equal. Again, with this derivation, the set B can be reduced to containing OFFCVs only.

Next, based on these derivations mentioned, we define *semi-critical summable* for a function.

Definition 5: A function f is said to be semi-critical k-summable if and only if for some k, $2 \leq k$, there exist two sets $A \subseteq f^{on}$, $B \subseteq f^{off}$ such that either $\{A^{(j)} | \boldsymbol{A^{(j)}}$ **is an ONCV**$\}$, $\{B^{(j)} | f(B^{(j)}) = 0\}$, or $\{A^{(j)} | f(A^{(j)}) = 1\}$, $\{B^{(j)} | \boldsymbol{B^{(j)}}$ **is an OFFCV**$\}$, and

$$\sum_{j=1}^{k} A^{(j)} = \sum_{j=1}^{k} B^{(j)} \tag{5}$$

provided that repetition of vectors in the sets A and B are permitted. If f is semi-critical k-summable for some k, f is said to be *semi-critical summable*.

Theorem 5 (Semi-Critical Summable Theorem): A positive unate function f is a non-TF if and only if f is semi-critical summable.

Proof: ($\Rightarrow$) If f is a non-TF, f is k-summable for some k by Corollary 1. Hence, there exist two sets A and B such that $\{p_1, p_2, ..., p_k\} \in A$, $\{q_1, q_2, ..., q_k\} \in B$, and the summations of vectors in the sets A and B are equal. Let p_i be a non-ONCV $\in f^{on}$ in A, then we can apply a decrement operation on one of its inputs. Assume that we apply a decrement operation on the m^{th} input of p_i. That is, the m^{th} input of p_i is 1, and the m^{th} input summation of vectors in $A \geq 1$. Since the summations of vectors in A and B are equal, the m^{th} input summation of vectors in B also ≥ 1. Thus, there exists a vector q_j in B whose m^{th} input is 1, and we can apply a decrement operation on the m^{th} input of q_j by Lemma 1. After applying a decrement operation on the m^{th} input of both vectors p_i and q_j, the summations of vectors in A and B are still equal.

Thus, we know that for any non-ONCVs in the set A, we can find a corresponding vector in the set B such that we can apply decrement operations on the same input of both vectors. According to Theorem 3, each non-ONCV in the set A can be finally turned into an ONCV. Therefore, we can apply a decrement operation on each non-ONCV in the set A and the corresponding vector in the set B until all the vectors in the set A are ONCVs. At this moment, all the vectors in the set B are still $\in f^{off}$. Thus, f is semi-critical summable by Definition 5.

Similarly, we can apply increment operations on the vectors in the sets A and B, and turn all the vectors in the set B into OFFCVs while the set A contains vectors $\in f^{on}$. Thus, f is semi-critical summable by Definition 5.

($\Leftarrow$) Since when f is semi-critical summable, f is summable definitely. From Corollary 1, we know that if f is summable, f is a non-TF. Thus, if f is semi-critical summable, f is a non-TF. ■

Theorem 5 indicates that if f is semi-critical summable, f is a non-TF. As compared with Corollary 1, the time complexity for checking non-TFs by using Theorem 5 is significantly reduced. Specifically, the time complexity for checking whether a function f, which contains s vectors $\in f^{on}$ and t vectors $\in f^{off}$, is k-summable is $O(s^k t^k)$, for $2 \leq k$. However, using Theorem 5, we do not need to check all the vector combinations in f^{on} and f^{off} simultaneously, Instead, we only need to check if there exist a set A consisting of ONCVs and a set B consisting of vectors $\in f^{off}$ such that the summations of vectors in the sets A and B are equal. Thus, the time complexity can be reduced to $O(|ONCV|^k t^k)$, where $|ONCV|$ is the number of ONCVs in f.

On the other hand, we only need to check if there exist a set A consisting of vectors $\in f^{on}$ and a set B consisting of OFFCVs such that the summations of vectors in the sets A and B are equal. Thus, the time complexity can also be reduced to $O(s^k |OFFCV|^k)$, where $|OFFCV|$ is the number of OFFCVs in f. Note that we cannot turn all the vectors in the set A into ONCVs and turn all the vectors in the set B into OFFCVs simultaneously. Thus, the time complexity for checking if f is k-summable will be $O(min\{(|ONCV|^k t^k), (s^k |OFFCV|^k)\})$.

For example, for a 7-input function $f = abcd+abcef+abdeg$, f has 12 vectors $\in f^{on}$ and 116 vectors $\in f^{off}$. The time complexity for checking f is k-summable is $O(12^k 116^k)$ originally. However, f has 3 ONCVs and 7 OFFCVs only. Hence, by Theorem 5, we can reduce the time complexity to either $O(3^k 116^k)$ or $O(12^k 7^k)$. Obviously, $O(12^k 7^k)$ is the better one.

TABLE I
NUMBERS OF FUNCTIONS IN DIFFERENT GROUPS OF INPUT N.

n	0	1	2	3	4	5	6
Number of NPN-equivalence classes of unate functions with n inputs (N_1)	1	1	1	3	11	95	8,170
Number of NPN-equivalence classes of TFs with n inputs (N_2)	1	1	1	3	9	48	504
Number of NPN-equivalence classes of non-TFs with n inputs (N_3)	0	0	0	0	2	47	7,666

IV. EXPERIMENTAL RESULTS

We implemented the proposed algorithm in C++ language. The experiments were conducted on an Intel Xeon E5-2650v2 2.60 GHz CentOS 6.7 platform with 64GBytes. Since the proposed Semi-Critical Summable Theorem focuses on the summation computation with ONCVs and OFFCVs of positive unate functions, we generate positive unate functions for each group of functions with the same number of inputs ranging from 5 to 9 as the benchmarks. We selected one positive unate function from each NPN-equivalence class.

TABLE I shows the number of functions in different groups of input. The numbers of NPN-equivalence classes of unate functions and TFs are provided from [13]. Thus, we can obtain the number of NPN-equivalence classes of non-TFs. Note that the numbers of NPN-equivalence classes of unate functions provided from [13] are only up to 6-input functions. Thus, we only have the exact numbers of functions and non-TFs ranging from 0 to 6 inputs. We have generated *all* the NPN-equivalent positive unate functions with 5 and 6 inputs, whose total numbers are 95 and 8170, respectively. For input numbers ranging from 7 to 9, we randomly picked 300,000 functions.

Originally, the summable checking procedure needs to compute all the combinations of the vectors $\in f^{on}$ and the vectors $\in f^{off}$. With the Semi-Critical Summable Theorem, we can either replace the combinations of vectors $\in f^{on}$ with ONCVs, or replace the combinations of vectors $\in f^{off}$ with OFFCVs. Thus, in the first experiment, the reduction of computations focuses on the difference between the number of vectors $\in f^{on}$ and that of ONCVs, or the difference between the number of vectors $\in f^{off}$ and that of OFFCVs.

TABLE II shows the average ratio $R1$ of the number of ONCVs to the number of vectors $\in f^{on}$ (s), and the average ratio $R2$ of the number of OFFCVs to the number of vectors $\in f^{off}$ (t) for all the generated functions, while $s+t = 2^n$. The average ratio is $R1 = 39.04\%$ and $R2 = 29.84\%$ when the number of inputs is 5, which is the highest value among all the results. As the number of inputs increases, $R1$ and $R2$ are reduced significantly. When the number of inputs is 9, $R1$ and $R2$ are

TABLE II
THE AVERAGE RATIOS OF THE NUMBER OF ONCVS TO THE NUMBER OF VECTORS $\in f^{on}$ (s) AND THE NUMBER OF OFFCVS TO THE NUMBER OF VECTORS $\in f^{off}$ (t), $s + t = 2^n$.

n	N_1	$R1 = \|ONCV\|/s$ (%)	$R2 = \|OFFCV\|/t$ (%)
5	95	39.04	29.84
6	8,170	28.51	26.63
7	300,000	19.77	14.75
8	300,000	9.57	7.06
9	300,000	6.51	3.85

TABLE III
THE COMPARISON OF THE AVERAGE NUMBER OF COMPUTATIONS BETWEEN THE ORIGINAL TIME COMPLEXITY AND THE REDUCED TIME COMPLEXITY.

n	N_1	$s \times t$	$min\{(\|ONCV\| \times t), (s \times \|OFFCV\|)\}$	Ratio (%)
5	95	227.31	63.26	27.83
6	8,170	970.00	230.26	23.74
7	300,000	3,549.17	506.81	14.28
8	300,000	12,232.06	835.46	6.83
9	300,000	36,652.27	1,348.48	3.68

smaller than 7% for the randomly generated 300,000 functions. Thus, using either ONCVs to replace the vectors $\in f^{on}$, or OFFCVs to replace the vectors $\in f^{off}$ improves the efficiency of summable checking procedure.

The second experiment shows the reduction of computations from the complexity viewpoint. The original time complexity of the k-summable checking procedure is $O(s^k t^k)$, and the reduced one is $O(min\{(|ONCV|^k t^k), (s^k |OFFCV|^k)\})$. Hence, we can extract and ignore the power of k, and compare the number of computations for the remaining terms only. TABLE III shows the comparison on the number of computations. As the number of inputs increases one, the number of vectors in f^{on} and f^{off} are roughly double. Thus, the product of $s \times t$ becomes about four times. Since our method can select the smaller part, which is either $|ONCV| \times t$ or $s \times |OFFCV|$, to compute, the number of computations will become about double or triple as the number of inputs increases one. In TABLE III, the largest computation ratio between our approach and the original complexity is only 27.83% for the 5-input functions, and this ratio keeps decreasing as the number of inputs increases. When the number of inputs is 9, we can reduce about 96% computations on average. Thus, the number of computations in the summable checking procedure using the proposed Semi-Critical Summable Theorem can be significantly reduced. When the power of k is further considered, the reduction of computations will be even tremendous.

V. CONCLUSION

Asummability is a sufficient and necessary condition for being a TF in Summable Theorem. However, the identification of TFs using Summable Theorem is not efficient in practice. Hence, in this work, we propose Semi-Critical Summable Theorem to deal with the high complexity issue in Summable Theorem. The experimental results show that the proposed theorems are sound, and advance the theoretic study about threshold functions.

REFERENCES

[1] V. Beiu, J. M. Quintana, and M. J. Avedillo, "VLSI implementations of threshold logic: A comprehensive survey," IEEE Trans. Neural Netw., vol. 14, no. 5, pp. 1217–1243, Sep. 2003.

[2] K. Berezowski and S. Vrudhula, "Automatic design of binary multiple-valued logic gates on the rtd series," *in Eight Euromicro Conf. on Digital System Design*, 2005.

[3] Y. Crama, and P. L. Hammer, "Boolean Functions: Theory, Algorithms, and Applications," Cambridge University Press, 2011.

[4] M. J. Ghazala, "Irredundant disjunctive and conjunctive forms of a Boolean function," I.B.M. Journal of Research and Development, vol. 1, pp. 171-176, April 1957.

[5] P.-Y. Kuo, C.-Y. Wang, and C.-Y. Huang, "On Rewiring and Simplification for Canonicity in Threshold Logic Circuits," in *Proc. ICCAD*, pp. 396-403, 2011.

[6] S.-Y. Lee, N.-Z. Lee, and J.-H. R. Jiang, "Canonicalization of threshold logic representation and its applications," in *Proc. ICCAD*, pp. 1-8, 2018.

[7] C.-C. Lin, C.-Y. Wang, Y.-C. Chen, and C.-Y. Huang, "Rewiring for Threshold Logic Circuit Minimization," in *Proc. DATE*, pp. 1-6, 2014.

[8] C.-C. Lin, C.-W. Huang, C.-Y. Wang and Y.-C. Chen, "In&Out: Restructuring for Threshold Logic Network Optimization," in *Proc. ISQED*, pp. 413-418, 2017.

[9] C.-C. Lin, C.-H. Liu, Y.-C. Chen, C.-Y. Wang, and S. Yamashita, "A New Necessary Condition for Threshold Function Identification," *IEEE Trans. on Computer-Aided Design*, pp. 5304-5308, vol. 39, No. 12, 2020.

[10] C.-H. Liu, C.-C. Lin, Y.-C. Chen, C.-C. Wu, C.-Y. Wang, and S. Yamashita, "Threshold Function Identification by Redundancy Removal and Comprehensive Weight Assignments," *IEEE Trans. on Computer-Aided Design*, vol. 38, no. 12, pp. 2284 - 2297, 2019.

[11] V. A. Mardiris, G. C. Sirakoulis, and I. G. Karafyllidis, "Automated design architecture for 1-D cellular automata using quantum cellular automata," *IEEE Trans. Computers*, vol. 64, no. 9, pp. 2476–2489, Sep. 2015.

[12] S. Minato, "Fast generation of prime-irredundant covers from binary decision diagrams," IEICE Trans. Fundamentals, vol. E76-A, no. 6, pp. 976-973, 1993.

[13] S. Muroga,"Threshold Logic and its Applications," New York, NY: John Wiley, 1971.

[14] A. Neutzling, J. M. Matos, A. I. Reis, R. P. Ribas, and A. Mishchenko, "Threshold logic synthesis based on cut pruning," in *Proc. ICCAD*, pp. 494-499, 2015.

[15] A. Neutzling, J. M. Matos, A. Mishchenko, A. I. Reis, and R. P. Ribas, "Effective Logic Synthesis for Threshold Logic Circuit Design," *IEEE Trans. on Computer-Aided Design*, vol. 38, no. 5, pp. 926-937, 2019.

[16] A. Neutzling, M. G. A. Martins, V. Callegaro, A. I. Reis, and R. P. Ribas, "A Simple and Effective Heuristic Method for Threshold Logic Identification," *IEEE Trans. on Computer-Aided Design*, vol. 37, no. 5, pp. 1023-1036, 2018.

[17] G. Papandroulidakis, A. Serb, A. khiat, G. Merreet, and T. Prodromakis, "Practical Implementation of Memristor-Based Threshold Logic Gates," *IEEE Trans. on Circuits and Systems*, vol. 66, no. 8, pp. 3041-3051, 2019.

[18] S. R. Petrick, "A direct determination of the irredundant forms of a Boolean function from a set of prime implicants," A.F. Cambridge Res. Center, Bedford, Mass., Report AFCRC-TR-56-110, 1956.

[19] W. V. Quine, "The problem of simplifying truth functions," Am. Math. Monthly, vol. 59, pp. 521-531, 1952.

[20] V. Saripalli, L. Liu, S. Datta, and V. Narayanan, "Energy-delay Performance of Nanoscale Transistors Exhibiting Single Electron Behavior and Associated Logic Circuits," *Journal of Low Power Electronics*, vol. 6, pp. 415-428, 2010.

[21] P. Wang, M. Y. Niamat, S. R. Vemuru, M. Alam, and T. Killian, "A Synthesis of Majority/Minority Logic Networks," *IEEE Trans. on Nanotechnology*, vol. 14, no. 3, pp. 473-483, 2015.

[22] R. O. Winder, "Enumeration of Seven-Argument Threshold Functions," *IEEE Trans. on Electronic Computers*, pp. 315-325, 1965.

[23] R. Zhang, P. Gupta, L. Zhong, and N. K. Jha, "Threshold Network Synthesis and Optimization and Its Application to Nanotechnologies," *IEEE Trans. Computer-Aided Design*, vol. 24, no. 1, pp.107-118, 2005.

[24] Y. Zheng, M. S. Hsiao, and C. Huang, "SAT-based Equivalence Checking of Threshold Logic Designs for Nanotechnologies," *in Proc. Great Lake Symp. VLSI*, May 2008, pp. 225-230.

[25] http://minisat.se/