An Effective and Efficient Heuristic for Rational-Weight Threshold Logic Gate Identification

Ting-Yu Yeh, Yueh Cho, and Yung-Chih Chen
Dept. of Electrical Engineering, National Taiwan University of Science and Technology
{m11007423, m11007429, ycchen.ee}@mail.ntust.edu.tw

***Abstract*—In CMOS-based current mode realization, the threshold logic gate (TLG) implementation with rational weights has been shown to be more cost-effective than the conventional TLG implementation without rational weights. The existing method for the rational-weight TLG identification is an integer linear programming (ILP)-based method, which could suffer from inefficiency for a Boolean function with a large number of inputs. This paper presents a heuristic for rational-weight TLG identification. We observe from the ILP solutions that in the ILP formulation, many variables related to the rational weights are redundant. Additionally, a rational-weight TLG could be transformed from a conventional TLG. Thus, the proposed method aims to identify the conventional TLG that can be transferred to a rational-weight TLG with lower implementation cost. We conducted the experiments on a set of TLGs with 4 $\sim$ 15 inputs. The results show that the proposed method has a competitive quality with an average ratio of 0.96, compared to the ILP-based method. Additionally, the proposed method spent only an average of approximately 2% of CPU time.**

***Index Terms*—Threshold logic, rational weights, logic synthesis.**

I. Introduction

Recent achievements in hardware realization of threshold logic and the rise of machine learning make threshold logic [1] re-attract enormous attention from researchers. Threshold logic is a more powerful representation than conventional Boolean logic composed of primitive gates. A threshold logic gate (TLG) can implement a complex Boolean function that a primitive gate cannot. In addition, due to the functional similarity between a TLG and an artificial neuron, threshold logic is regarded as a feasible solution for hardware implementation of artificial neural networks.

The Boolean function f of a TLG with n Boolean inputs, $x_1 \sim x_n$, is defined by (1). Each input x_i is accompanied with an integer weight w_i, and there is an integer threshold value w_T. The function evaluates to 1 if the sum of all the activated weights (i.e., $x_i = 1$) is greater than or equal to w_T; otherwise, it evaluates to 0.

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

For example, Fig. 1(a) shows the graphical representation of a TLG that has 4 inputs with weights -1, 1, 2, and 3, and a threshold value 4. When the input pattern is 0101, it evaluates to 1 because $1+3 = 4 \geq 4$. However, if the input pattern is 1110, the output is 0, since $(-1)+1+2 = 2 < 4$. A TLG can also be represented by a weight-threshold vector $[w_1, w_2, ..., w_n; w_T]$, e.g., $[-1, 1, 2, 3; 4]$ for the TLG.

This work was supported by the National Science and Technology Council, Taiwan, under grant MOST 111-2221-E-011-137-MY3.

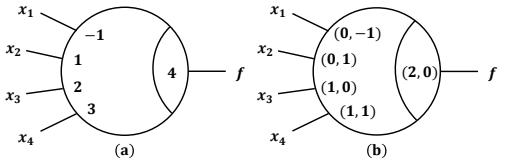


Fig. 1. (a) Conventional TLG, $f = [-1, 1, 2, 3; 4]$ and (b) R-TLG of $l = 2$, $f = [0, -1, 0, 1, 1, 0, 1, 1; 2, 0]$.

In recent years, Mozaffari *et al.* [2] proposed a novel CMOS-based current mode implementation of a TLG with **rational** weights, increasing the flexibility of TLG implementation. In the new implementation, an input x_i has l rational weights, $w_i^1 \sim w_i^l$, and there are l threshold weights, $w_T^1 \sim w_T^l$. The function of the rational-weight TLG is defined as (2).

$$f(x_1, x_2, ..., x_n) = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} ((\sum_{j=1}^{l} \frac{1}{j} w_i^j) * x_i) \geq \sum_{j=1}^{l} \frac{1}{j} w_T^j \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

For an activated rational weight w_i^j, the value it contributes to the weight sum is $\frac{1}{j} w_i^j$. Additionally, a threshold weight w_T^j contributes $\frac{1}{j} w_T^j$ to the overall threshold value. For clarity, we call a rational-weight TLG an **R-TLG** and a conventional TLG, which has no rational weight, a **TLG**. Furthermore, for an R-TLG, we call the rational weights w_i^j with $j > 1$ **true rational weights**, and the rational weights w_i^1 **false rational weights** since the values they contribute to the weight sum are integers.

Fig. 1(b) shows the R-TLG implementation of $l = 2$ for the same function in Fig. 1(a). Each input x_i has two rational weights, w_i^1 and w_i^2, e.g., $(w_1^1, w_1^2) = (0, -1)$ for x_1. Additionally, there are two threshold weights, $(w_T^1, w_T^2) = (2, 0)$. The R-TLG can be represented by the weight-threshold vector $[w_1^1, w_1^2, w_2^1, w_2^2, w_3^1, w_3^2, w_4^1, w_4^2; w_T^1, w_T^2] = [0, -1, 0, 1, 1, 0, 1, 1; 2, 0]$.

In [2], Mozaffari *et al.* also presented an integer linear programming (ILP)-based method for R-TLG identification of a given Boolean function, under a predetermined l. Compared with a TLG, an R-TLG could have a lower hardware implementation cost for the same Boolean function. Although the ILP-based identification method is effective, it could suffer from inefficiency for a large Boolean function or a large l.

978-3-9819263-7-8/DATE23/© 2023 EDAA

To solve the issue, we propose a heuristic method for R-TLG identification without ILP solving.

Given a TLG and a user-defined l, our objective is to find an R-TLG implementation that has a lower implementation cost. By analyzing the ILP results, we observe that many variables related to the rational weights in the ILP formulations are redundant, i.e., they are assigned 0. Additionally, we observe that a TLG can be directly transformed into an R-TLG. Thus, we first propose an approach for transforming a TLG into an R-TLG. Then, we adapt the heuristic TLG identification method [3] to find a TLG implementation that can lead to a lower-cost R-TLG implementation after transformation.

We conducted the experiments on a set of TLGs collected from the threshold logic networks (TLNs) generated by using the LUT-based synthesis method [4]. The experimental results show that the proposed method has a competitive quality with an average ratio of 0.9625, compared to the ILP-based method. Additionally, the proposed method is much more efficient. It can save an average of 98.24% of CPU time.

The remainder of this paper is organized as follows: Section II reviews some background on threshold logic, the heuristic method for TLG identification [3] and the ILP-based method for R-TLG identification [2]. Section III shows our three observations on R-TLG implementation, which inspire the proposed method to be presented in Section IV. Section V shows the experimental results. Finally, the conclusion is presented in Section VI.

II. Preliminaries

A. Background

Threshold logic is an alternative and more powerful representation of conventional Boolean logic. A complex Boolean function can be represented with only one TLG. For example, the function $f = x_1{}'x_2x_4 + x_3x_4$ can be implemented with only one TLG as shown in Fig. 1(a). A Boolean function $f(x_1, ..., x_n)$ is said to be a threshold function (TF), if it can be implemented with only one TLG. Unateness is a necessary condition of a TF. All TFs must be unate, but not necessarily vice versa. For example, $g = x_1x_2 + x_3x_4$ is a unate function, but not a TF.

A TLN is a Boolean logic network composed of TLGs. In general, the weights and the threshold value in a TLG are integers, which can be positive or negative. A negative weight can be converted to a positive weight through the *negation property* [5]. Let $f(x_1, ..., x_n)$ be $[w_1, w_2, ..., w_i, ...w_n; w_T]$. The negation of x_i, i.e., $x_i \rightarrow x_i{}'$, changes the weight-threshold vector to $[w_1, w_2, ..., -w_i, ...w_n; w_T - w_i]$. Take the function f in Fig. 1(a) as an example, we can negate x_1 and convert the weight-threshold vector $[-1, 1, 2, 3; 4]$ into $[1, 1, 2, 3; 5]$.

In this work, we pre-convert all the negative weights (if any) to be positive before we manipulate a TLG, and then convert them back when we evaluate its hardware implementation cost.

B. Design Automation for Threshold Logic

In the past few decades, many design automation techniques for threshold logic have been proposed. Most of them are dedicated to conventional threshold logic. The TLG identification technique checks whether a Boolean function can be represented with a TLG and computes the weights and threshold value. Several exact and heuristic methods were proposed [3], [6]–[9]. The exact method based on ILP [6] can compute the minimum weights and threshold value. The heuristic methods [3], [8], [9] have a competitive quality and are more efficient, compared to the exact ILP-based method.

The synthesis technique maps a Boolean logic network into a TLN [4], [6], [7]. The state-of-the-art synthesis method [4] is a lookup table (LUT)-based approach, which takes advantage of the LUT-based mapping technology for Field Programmable Gate Arrays (FPGAs) [10] and is very effective in minimizing the TLG count and logic depth. Furthermore, the optimization technique minimizes the hardware implementation cost of a TLN [11], [12].

Few techniques exist for rational-weight threshold logic. In [2], Mozaffari *et al.* presented a CMOS-based hardware implementation [13] for R-TLGs and an ILP-based method for R-TLG identification. The authors also demonstrated that for some functions, an R-TLG has a lower implementation cost than a TLG. Although the ILP-based identification method is effective, it could suffer from the scalability issue. Thus, in this work, we propose a heuristic method to compute the lower-cost R-TLG for a TLG.

C. Hardware Implementation of TLG and R-TLG

S. Bobba and I. N. Hajj [13] proposed a CMOS-based current-mode implementation for a TLG. The implementation consists of three parts: two differential parts and one sensor part. One differential part is to implement the positive input weights and the negative threshold value, and the other differential part is to implement the negative input weights and the positive threshold value. The sensor part compares the currents from the two differential parts to determine the output value.

The hardware implementation cost of a TLG is estimated based on the area of the transistors in the differential parts. It is defined by (3), which is the sum of the absolute values of weights and threshold value. For example, the cost of the TLG in Fig. 1(a) is 11 $(= |-1| + 1 + 2 + 3 + 4)$.

$$TLG\ cost = \sum_{i=1}^{n}(|w_i|) + |w_T| \tag{3}$$

In [2], Mozaffari *et al.* adapted the CMOS-based current-mode implementation for an R-TLG. In the differential parts, a rational weight w_i^j is implemented with j transistors connected in series. Thus, the hardware implementation cost of w_i^j is j times that of w_i^1. The cost of an R-TLG is defined by (4). For example, the cost of the R-TLG in Fig. 1(b) is 10 $(= 2 * |-1| + 2 * 1 + 1 + 1 + 2 * 1 + 2)$.

$$R-TLG\ cost = \sum_{i=1}^{n}(\sum_{j=1}^{l}(j * |w_i^j|)) + \sum_{j=1}^{l}(j * |w_T^j|) \tag{4}$$

These two examples also demonstrate that an R-TLG can have a lower cost than a TLG for implementing the same function. For more details on the hardware implementation, please refer to [2], [13].

D. Heuristic for TF identification

In [3], Liu *et al.* proposed an efficient and effective heuristic method for TF identification. The key procedure consists of two stages. First, it constructs a system of inequalities according to the given Boolean function. The inequalities are in fact constraints on the relationships among the weights. Second, it finds a weight assignment that satisfies all the inequalities, and computes the threshold value.

Let us use an example taken from [3] to illustrate the TF identification process. Suppose that the given function is $f = x_1x_2 + x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_1x_5x_6$. The method first checks whether f is unate or not, since only a unate function can be a TF. Next, it computes the weight ordering of f, which is $w_1 > w_2 > w_3 = w_4 > w_5 = w_6$, based on the modified Chow's parameters [5].

Then, the method finds the ON-set cubes and the OFF-set cubes of f, which are $\{x_1x_2, x_1x_3, x_1x_4, x_2x_3, x_2x_4, x_1x_5x_6\}$ and $\{x_1'x_2', x_1'x_3'x_4', x_2'x_3'x_4'x_5', x_2'x_3'x_4'x_6'\}$, respectively. Since the weight sum of an ON-set cube is always larger than that of an OFF-set cube, the method pairs the two sets of cubes to construct the system of inequalities and eliminates redundant inequalities based on the weight ordering. Additionally, the method simplifies the inequality system by using the same symbol to replace the equal weights in the inequalities according to the weight ordering. For example, w_1 is replaced by A, w_2 is replaced by B, w_3 and w_4 are replaced by C, and w_5 and w_6 are replaced by D.

After that, the method conducts the weight assignment procedure to find the smallest weight values satisfying all the inequalities. It starts with an initial weight assignment, in which each weight is assigned the smallest positive integer that satisfies the weight ordering, e.g., $A = 4$, $B = 3$, $C = 2$, and $D = 1$. In each iteration, the current weight assignment is used to check whether the inequality system is satisfied. If there are unsatisfied inequalities, called false inequalities, the method finds critical weights (CWs) and increases their values. The CW is a weight that appears more times on the greater side than the lesser side in the false inequalities. Increasing the values of CWs could cause more false inequalities to be satisfied. The weight adjustment repeats until all the inequalities are satisfied or a termination condition is reached.

Finally, if a legal weight assignment is found, f is a TF, and then the method computes the threshold value which is the largest weight sum among the OFF-set cubes plus 1.

In this work, we adapted the heuristic method to identify R-TLGs. Unlike [3], which aims to find a TLG with minimum weights and threshold value, our objective is to find a TLG that has a lower cost after it is transformed into an R-TLG.

E. ILP-based R-TLG Identification

In [2], Mozaffari *et al.* presented an ILP-based method for R-TLG identification. The method models the identification problem as an ILP problem. Given a Boolean function f, to construct the ILP formulation, the method first enumerates the truth table of f. Then, for each input pattern, it generates an inequality (i.e., a constraint) based on (2). The objective function of the ILP formulation is to minimize the hardware cost of the R-TLG defined by (4). If the formulation has a solution, f has an R-TLG implementation and its rational weights leading to the minimum cost are obtained.

According to [2], for an n-input Boolean function, there are $2^n + l*3*(n+1)$ constraints and $l*3*(n+1)$ variables in the ILP formulation for finding an R-TLG implementation with l rational weights. Although the ILP-based is effective, it could suffer from inefficiency for a large n or l. Thus, in this work, we are going to propose an efficient heuristic method that has a competitive quality with the ILP-based method.

TABLE I
PERCENTAGES OF TRUE R-WEIGHTS WHOSE VALUES ARE 0

l	2	3	4	5	6	7	avg.
0-value true R-weights (%)	65.71	76.90	84.72	88.64	90.90	92.39	84.69

III. OBSERVATIONS

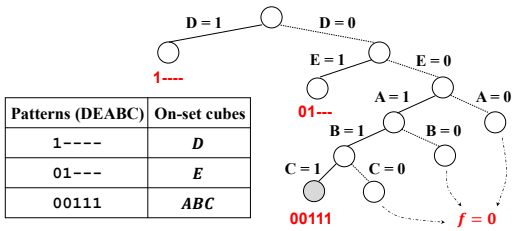
In this section, we present three observations on the ILP-based R-TLG identification [2] and the transformation from a TLG to an R-TLG, which inspire the proposed method.

Observation 1: Most of the true rational weights are 0. In ILP-based R-TLG identification, the number of variables in the ILP formulation dramatically increases, as the input number of the given Boolean function, n, and the number of rational weights, l, increase. However, we observe that most of the true rational weights reported by the ILP-based method are 0. That is, the corresponding variables are redundant and can be removed from the ILP formulation without affecting the quality. Table I shows the percentages of 0-value true rational weights in the R-TLGs with $4 \sim 15$ inputs for different l. The benchmark functions are collected from the TLNs generated by using the LUT-based synthesis method [4] and the R-TLGs are computed by the ILP-based method. As you can see, most of the true rational weights have a value of 0 and the percentage increases as l increases. Thus, it is inefficient to consider all the variables when we identify an R-TLG.

Observation 2: Constraining all the true rational weights to 0, 1, or -1 does not affect the quality. In general, a rational weight can be an arbitrary integer and its value represents the hardware configuration. We observe that, in the experiment of Table I, when we constrain all the true rational weights to 0, 1, or -1, the quality in terms of R-TLG cost does not change.

Observation 3: A TLG can be transformed into an R-TLG. We observe that a TLG can be directly transformed into an R-TLG with the following procedure: Given a TLG and the value of rational weight count, l, we first divide each weight and the threshold value by l to obtain a new weight-threshold vector. Although some weights and the threshold value may become fractions, the function remains the same. Then, for each fraction, its integer part is implemented with the false rational weight and its fractional part is realized with a combination of the other true rational weights.

According to the three observations, the objective of the proposed method is to find a TLG which can be transformed



Patterns (DEABC)	On-set cubes
1----	*D*
01---	*E*
00111	*ABC*

Fig. 2. Example of computing the ON-set cubes.

into an R-TLG with a lower implementation cost. When we find a TLG, we do not consider the variables corresponding to the rational weights. Additionally, we only find the weights that can result in the 1-value or -1-value rational weights.

IV. Proposed Method

The inputs of the problem under consideration are a Boolean function with its TLG implementation and a user-defined l. Our objective is to identify the R-TLG implementation with l of the function with a lower hardware cost.

The proposed method aims to be more efficient in identifying R-TLG. It contains four stages: create the inequalities system, build the tables for transformation, search the R-TLG weight assignment, and refine the weight assignment for minimizing cost.

A. Generation of the System of Inequalities

We directly use $f'(A,B,C,D,E) = [1,1,1,-3,-3;-3]$ as an example to demonstrate the proposed method. First, all the negative weights are converted to be positive through the negation property. As a result, we can get $f = [1,1,1,3,3;3]$ and the weight ordering is $D = E > A = B = C$.

Then, we compute the ON-set cubes and the OFF-set cubes of f by implicitly building a binary decision tree. The variable order of the decision tree is determined according to the weight ordering.

To compute the ON-set cubes, at each iteration, we first assign 1 to the target variable, and then assign 0. When the assigned value is 1, we check whether the current assignment can directly determine $f = 1$ regardless of other unassigned values. If yes, an ON-set cube is obtained by collecting the variables that are assigned 1. On the contrary, when the assigned value is 0, we check whether $f = 0$ can be directly determined. If yes, we fathom the node. Fig. 2 shows the binary decision tree and the computed ON-set cubes.

The method of computing the OFF-set cubes is similar to that of computing the ON-set cubes, while the considered values are opposite. We first assign 0 to the target variable, and then 1. When the assigned value is 0, we check whether the current assignment can directly determine $f = 0$ regardless of other unassigned values. If yes, an OFF-set cube is obtained by collecting the variables that are assigned 0. Similarly, when we assign 1 to the target variable, we check whether $f = 1$ can be directly determined. If yes, we fathom the node. The computed OFF-set cubes of f are $\{A'D'E', B'D'E', C'D'E'\}$.

TABLE II
System of inequalities of f'

Inequalities							
1.	D	$>$	$B+C$	4.	E	$>$	$B+C$
2.	D	$>$	$A+C$	5.	E	$>$	$A+C$
3.	D	$>$	$A+B$	6.	E	$>$	$A+B$

After obtaining the ON-set cubes and the OFF-set cubes, we use the method in [3] to generate the system of inequalities and simplify them as shown in Table II.

B. Building of the Tables for Rational Weight Transformation

Based on our observations, we first define a *single-device* table that constrains the values of the true rational weights to be 0 or 1. It records all the weights within a certain range that can be implemented under the constraint and the corresponding minimum hardware costs. Constructing the table early can avoid repeated calculation.

We also define a *multiple-device* table without any restriction on the rational weight values. The table is used to obtain a lower hardware cost during the adjustment process of the weights and the threshold value.

Algorithm 1 Build the single-device table

Input: The value of l and a upper bound b
Output: The single-device table T_s

1: $T_s = \emptyset$;
2: $Lcm \leftarrow$ LeastCommonMultiple(1, ..., l);
3: $T_s \leftarrow$ CreateFundamentalWeights(l, Lcm);
4: **for** $i = 1$ to b **do**
5: $pairSet$ = $\emptyset$;
6: $pairSet \leftarrow$ EnumerateAllPairCombinations(i, T_s);
7: **for** each pair combination p in $pairSet$ **do**
8: **if** (SingleDevice(p)) **then**
9: c = CalculateCost(p);
10: **if** (Entry i does not exist in T_s) **then**
11: Insert $[i, c]$ into T_s;
12: **else if** (c is lower than the cost in T_s) **then**
13: Update $[i, c]$ in T_s;
14: **return** T_s

Algorithm 1 shows the proposed method for constructing the single-device table T_s. We first calculate the least common multiple Lcm among all the integers from 1 to l, according to the given value of l (Line 2). The weights contributed by different rational weights are multiplied by Lcm to establish the fundamental weights (Line 3).

For example, if $l = 3$, the value of Lcm is 6 and we use the rational weight (w_i^1, w_i^2, w_i^3) to create the table. The fundamental weights $6 * (\frac{1}{1}, \frac{1}{2}, \frac{1}{3}) = (6, 3, 2)$ are created and the corresponding costs are $(1, 2, 3)$, respectively.

Then, for all the possible weight values from 1 to the upper bound b, we iteratively consider one value i (Line 4). At each iteration, we first find all the pair combinations from T_s, which can form i (Lines 5 and 6). If a combination p can be implemented by the true rational weights with a value 0 or 1

TABLE III
EXAMPLE OF T_s AND T_m WITH $l = 3$

T_s [weight, cost]			T_m [weight, cost]			
[2, 3]	[11, 6]	[20, 6]	[2, 3]	[8, 4]	[14, 5]	[20, 6]
[3, 2]	[12, 2]	[21, 5]	[3, 2]	[9, 3]	[15, 4]	[21, 5]
[5, 5]	[14, 5]	[23, 8]	[4, 6]	[10, 7]	[16, 8]	[22, 9]
[6, 1]	[15, 4]	[24, 4]	[5, 5]	[11, 6]	[17, 7]	[23, 8]
[8, 4]	[17, 7]		[6, 1]	[12, 2]	[18, 3]	[24, 4]
[9, 3]	[18, 3]		[7, 8]	[13, 9]	[19, 10]	

(Line 8), we compute the cost of p, denoted as c (Line 9). Then, we check whether the entry i exists in T_s or not (Line 10). If not, we insert the entry i with the cost c, $[i, c]$, into T_s (Line 11). Otherwise, if c is lower than the cost stored in T_s, we update $[i, c]$ in T_s (Lines 12 and 13).

The method of constructing the multiple-device table T_m is similar to that of T_s, except Line 8 in Algorithm 1 is not necessary. Table III shows the T_s and T_m under $l = 3$ and $b = 24$.

C. Weight Adjustment

The proposed method basically adjusts the selected weights up by one level based on the single-device table T_s.

Since the initial weight assignment, i.e., the given TLG, might not be implementable according to T_s, we first adjust the weights to a closet greater weight value in T_s, which satisfies the weight ordering.

Next, we iteratively select weights to adjust based on two methods. If the current weight assignment does not satisfy the system of inequalities, i.e., illegal assignment, we use **Method 1** to select all the CWs as the previous method [3]. Otherwise, we use **Method 2** to select the weight that leads to the largest cost reduction when adjusted. If there is more than one weight, we choose the largest weight. For Method 2, we adjust the selected weight and the other weights that are identical to it in the weight ordering (if any) simultaneously.

At each iteration, after adjustment, if it causes the weight ordering to be violated, we also adjust the other weights to meet the ordering. Furthermore, if the current weight assignment is legal, we check whether the possible threshold value is implementable or not according to T_m. The possible threshold value is within the range between the smallest sum of the ON-set cubes and the largest sum of the OFF-set cubes plus 1. If all the values within the range after conversion with negation property are not implementable according to T_m and none of them is 0, the current weight assignment is seen as an illegal assignment as well.

If the weight assignment is legal, we compute its cost based on T_m and the threshold value is determined as the implementable value within the range and having the lowest cost. Please note that the negative variables need to be restored with the negative property when we compute the cost. Additionally, if its cost is lower than that of the last legal weight assignment, we add it into the *downhill set*. If its cost is lowest among all the legal weight assignments so far, we store it in the *best set*. We also estimate the *most optimistic cost* of the legal weight

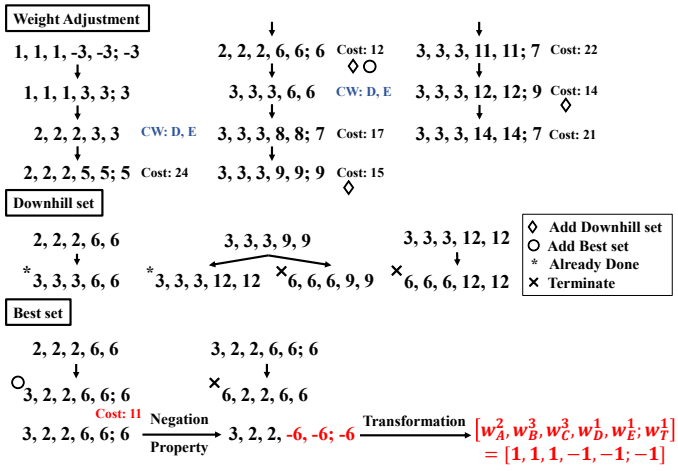


Fig. 3. Details of the weight adjustment and the refinement processes for $f' = [1, 1, 1, -3, -3; -3]$.

assignment based on (5). It is the possible lowest cost that can be achieved in the following adjustments.

The adjustment process terminates when the most optimistic cost of the current legal weight assignment is larger than that of the best set.

$$Cost = \sum_{i=1}^{n}(|w_i|/Lcm + 1) + |w_T|/Lcm + 1 \qquad (5)$$

Fig. 3 shows the details of the weight adjustment process for the example of $f = [1, 1, 1, 3, 3; 3]$. The collected weight assignments in the downhill set are $[2, 2, 2, 6, 6]$, $[3, 3, 3, 9, 9]$, and $[3, 3, 3, 12, 12]$. The best set has only one weight assignment $[2, 2, 2, 6, 6]$ with a threshold value of 6, and its cost is 12. The process terminates when the weight assignment becomes $[3, 3, 3, 14, 14]$, for which the most optimistic cost is 13 under the threshold value of 7, and it is larger than the cost of the best set.

D. Refinement

In this stage, we would like to refine the weight assignments in the downhill set and the best set to find a better weight assignment that has a lower cost.

We first tackle the downhill set. For each weight assignment in the set, we consider every weight group, in which the weights are equal in the weight ordering, one at a time. We simultaneously adjust the weights in the group up to the closet lower-cost weight value (if any) in T_s. If the new weight assignment is legal and has a lower cost, we add it into the downhill set. Furthermore, if the new cost is lower than that of the best set, we update the best set. The process terminates when no new weight assignment exists in the downhill set.

For example, for $[3, 3, 3, 9, 9]$ in the downhill set, we separately adjust the first three weights of 3 to 6 and the last two weights of 9 to 12 to obtain the two weight assignments $[6, 6, 6, 9, 9]$ and $[3, 3, 3, 12, 12]$. Among them, the second weight assignment $[3, 3, 3, 12, 12]$ is legal and has a lower cost than $[3, 3, 3, 9, 9]$, and, thus, we add it to the downhill set. Additionally, the best set does not need to be updated.

Second, we deal with the best set. The refinement method for the best set is similar to that for the downhill set. The main difference is that when we consider a weight group, we only adjust one weight that leads to the largest cost reduction, rather than all the weights in the group. For example, for the weight assignment $[2, 2, 2, 6, 6]$, when we consider the first weight group, we adjust the first weight to 3 to generate the new weight assignment $[3, 2, 2, 6, 6]$. The new weight assignment is legal and has a lower cost, and, thus, we add it into the best set. Again, the process terminates when no new weight assignment exists in the best set.

Finally, we can obtain the lowest-cost weight assignment from the best set and transform it into an R-TLG based on T_m. In the example, it is $[3, 2, 2, 6, 6]$ and the threshold value is 6. After restoring the negative weights through the negation property, we obtain $f' = [3, 2, 2, -6, -6; -6]$. It then can be transformed to an R-TLG $[w_A^2, w_B^3, w_C^3, w_D^1, w_E^1; w_T^1] = [1, 1, 1, -1, -1; -1]$ and the cost is 11.

Fig. 3 also summarizes the refinement process for the function f'.

V. Experimental Results

We implemented the proposed method in C++ language and conducted the experiments on a Linux platform with the Intel Core i5-10500 processor and 8GB memory. The ILP solver is *GUROBI* [14]. The TLG benchmarks are collected from the TLNs generated by the synthesis method [4], in which each TLG has at most 15 inputs. We remove the duplicate TLGs and cluster them according to their input counts. In the experiments, each TLG is transformed into an R-TLG by separately using our method and the ILP-based method [2].

The experimental results with $l = 2$ and $b = 400$ are summarized in Table IV. Columns 1 and 2 show the TLG sets of different input counts and the numbers of TLGs in each TLG set. Columns $3 \sim 5$ show the results of our method. They are the average percentages of implementation cost reduction compared with the given TLGs (R), the average CPU time AT for processing a TLG, and the total CPU time TT for processing all the TLGs. The corresponding results of the ILP-based method are shown in Columns $6 \sim 8$. The last two columns show the ratios of the results of our method and the ILP-based method. The results show that our method saves the cost with an average ratio of 0.9956, compared to the ILP-based method. Additionally, our method spent only an average of approximately 0.23% of CPU time.

Table V shows the ratios of the results of our method and the ILP-based method for different l. It depicts that our method has a competitive capability for saving the cost with an average ratio of 96.25%, and it only needs 1.76% of CPU time, compared with the ILP-based method.

VI. Conclusion

In this work, we propose an effective and efficient heuristic method for R-TLG identification. We first present three observations on R-TLG implementation. Then, we adapt the heuristic method for TLG identification to find a TLG implementation that can be transformed to a lower-cost R-TLG based on the observations. The experimental results show that the proposed method is more efficient than the ILP-based method, and has a competitive quality.

TABLE IV
Comparison of our method with the ILP-based method [2] under $l = 2$

TLG		Ours			ILP			Ratio (Ours/ILP)	
n	$\#G$	R (%)	AT (ms)	TT (ms)	R (%)	AT (ms)	TT (ms)	R	TT
4	102	21.78	0.01	1.48	21.78	4.01	409	1.0000	0.0035
5	347	30.73	0.02	8.03	31.22	4.52	1570	0.9843	0.0051
6	790	38.33	0.03	27.47	38.69	7.46	5891	0.9907	0.0047
7	1003	44.92	0.05	51.82	45.18	16.18	16233	0.9942	0.0032
8	1071	49.32	0.07	75.38	49.52	31.71	33965	0.9959	0.0022
9	681	52.35	0.09	61.46	52.51	55.38	37714	0.9970	0.0016
10	431	53.38	0.14	60.93	53.54	108.57	46791	0.9970	0.0013
11	280	53.98	0.37	103.14	54.13	229.96	64388	0.9972	0.0016
12	173	53.74	0.78	134.25	53.94	490.57	84868	0.9964	0.0016
13	101	53.66	3.31	334.50	53.80	1173.84	118558	0.9973	0.0028
14	80	55.69	0.79	63.37	55.75	2573.76	205901	0.9989	0.0003
15	51	56.23	0.20	10.25	56.34	7014.49	357739	0.9981	0.0003
Avg.		47.01	0.49		47.20	975.87		0.9956	0.0023

TABLE V
Comparison of our method with the ILP-based method

Value of l		2	3	4	5	6	7	Avg.
Upper bound b		400	800	1000	5000	5000	20000	–
Ratio (Ours/ILP) (%)	Cost	99.56	95.17	95.97	96.29	96	94.49	96.25
	Time	0.23	0.29	0.62	1.39	2.32	5.7	1.76

References

[1] R. O. Winder, "Single stage threshold logic," in *Proc. the Second Annual Symp. on Switching Circuit Theory and Logical Design*, pp. 321–332, 1961.

[2] S. N. Mozaffari *et al.*, "A generalized approach to implement efficient cmos-based threshold logic functions," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, pp. 946–959, March 2018.

[3] C.-H. Liu *et al.*, "Threshold function identification by redundancy removal and comprehensive weight assignments," *IEEE Trans. Computer-Aided Design*, vol. 38, no. 12, pp. 2284–2297, Dec. 2019.

[4] A. Neutzling *et al.*, "Effective logic synthesis for threshold logic circuit design," *IEEE Trans. Computer-Aided Design*, vol. 38, pp. 926–937, May 2019.

[5] S. Muroga, *Threshold logic and its applications*. New York, NY: John Wiley, 1971.

[6] R. Zhang *et al.*, "Threshold network synthesis and optimization and its application to nanotechnologies," *IEEE Trans. Computer-Aided Design*, vol. 24, pp. 107–118, Jan. 2005.

[7] T. Gowda *et al.*, "Identification of threshold functions and synthesis of threshold networks," *IEEE Trans. Computer-Aided Design*, vol. 30, pp. 665–677, May 2011.

[8] A. Neutzling *et al.*, "A simple and effective heuristic method for threshold logic identification," *IEEE Trans. Computer-Aided Design*, vol. 37, pp. 1023–1036, Jul. 2017.

[9] C.-C. Lin *et al.*, "A new necessary condition for threshold function identification," *IEEE Trans. Computer-Aided Design*, vol. 39, no. 12, pp. 5304–5308, Dec. 2020.

[10] Berkeley logic synthesis and verification group, "ABC: A system for sequential synthesis and verification, release 70930," 2007.

[11] Y.-C. Chen *et al.*, "Fast synthesis of threshold logic networks with optimization," in *Proc. Asia South Pacific Design Automation Conf.*, pp. 486–491, 2016.

[12] N.-Z. Lee and J.-H. R. Jiang, "Constraint solving for synthesis and verification of threshold logic circuits," *IEEE Trans. Computer-Aided Design*, vol. 40, no. 5, pp. 904–917, May 2021.

[13] S. Bobba and I. N. Hajj, "Current-mode threshold logic gates," *Proc. Int. Conf. on Computer Design*, pp. 235–240, 2000.

[14] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2022.