

A Novel Graphical Technique for Combinational Logic Representation and Optimization

Vedhas Pandit, Björn Schuller

Angaben zur Veröffentlichung / Publication details:

Pandit, Vedhas, and Björn Schuller. 2017. "A Novel Graphical Technique for Combinational Logic Representation and Optimization." *Complexity* 27: 9696342-1-9696342-12.
<https://doi.org/10.1155/2017/9696342>.

Nutzungsbedingungen / Terms of use:

CC BY 4.0

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:
CC-BY 4.0: Creative Commons: Namensnennung
Weitere Informationen finden Sie unter: / For more information see:
<https://creativecommons.org/licenses/by/4.0/deed.de>



Research Article

A Novel Graphical Technique for Combinational Logic Representation and Optimization

Vedhas Pandit¹ and Björn Schuller^{1,2}

¹*Chair of Embedded Intelligence for Health Care and Wellbeing, University of Augsburg, Augsburg, Germany*

²*Group on Language, Audio & Music (GLAM), Imperial College London, London, UK*

Correspondence should be addressed to Vedhas Pandit; vedhas@gmail.com

Received 5 June 2017; Revised 11 October 2017; Accepted 14 November 2017; Published 31 December 2017

Academic Editor: Michele Scarpiniti

Copyright © 2017 Vedhas Pandit and Björn Schuller. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We present a new technique for defining, analysing, and simplifying digital functions, through hand-calculations, easily demonstrable therefore in the classrooms. It can be extended to represent discrete systems beyond the Boolean logic. The method is graphical in nature and provides complete “implementation-free” description of the logical functions, similar to binary decision diagrams (BDDs) and Karnaugh-maps (K-maps). Transforming a function into the proposed representations (also the inverse) is a very intuitive process, easy enough that a person can hand-calculate these transformations. The algorithmic nature allows for its computing-based implementations. Because the proposed technique effectively transforms a function into a scatter plot, it is possible to represent multiple functions simultaneously. Usability of the method, therefore, is constrained neither by the number of inputs of the function nor by its outputs in theory. This, being a new paradigm, offers a lot of scope for further research. Here, we put forward a few of the strategies invented so far for using the proposed representation for simplifying the logic functions. Finally, we present extensions of the method: one that extends its applicability to multivalued discrete systems beyond Boolean functions and the other that represents the variants in terms of the coordinate system in use.

1. Introduction and Literature Overview

Combinational logic optimization is one of the first steps when designing any digital circuit. This practice helps chip designer save on number of transistors, chip area, and helps reduce logic delays and power requirements. It is no surprise therefore that many efforts have been made to develop fully functional, interactive programs for the industry even in the 80s, such as MIS [1] and SOCRATES [2] and the field has only grown ever since with the advent of many companies competing in this domain. In this section, we discuss various logic optimization methods. Advantages and drawbacks associated with each of these popular methods, especially the ones that offer graphical representation of a function or are simple enough for humans to hand-calculate, is the primary focus of this section. In view of the listed

advantages and drawbacks, characteristic features of an ideal methodology are then presented.

A Karnaugh-map (K-map) [3, 4] is a graphical technique for combinational logic optimization, which offers a very intuitive way to hand-calculate the proposed graphical representation of a Boolean function and its reduction. Using different colour schemes, multiple output functions involving the same inputs can be represented on a single K-map. However, it is only ideal for a function where the number of inputs is less than five. To represent functions involving more than four variables, one of the proposed implementations splits the graph into subgraphs—each subgraph representing only four variables at a time [5]. Another extension of the K-maps does not require splitting the graph into smaller units to represent functions with more than four variables, but some of the variable inversions can no longer remain adjacent to each other [6]—like in the case of K-maps for four inputs.

A binary decision diagram (BDD) [7–10] is also a graphical method that is easy enough to implement and visualise. BDDs are often put to use as a data structure to both represent the Boolean functions and to perform the operations efficiently. Unlike K-maps, usability of this approach is not constrained by the number of inputs. Optimization is primarily about choosing the correct ordering of input variables and about reducing the number of “nodes” by merging equivalent nodes and by removing redundant ones. Diagram complexity depends a lot on the ordering of the input variables and the respectively applied decomposition type [7, 11], and it is not always easy to devise the correct order by mere visual inspection. This however, to a limited extent, is also a downside of the graphical approaches presented in this paper. As for BDDs, moreover, one cannot translate a given graphical representation back to its Boolean expression just by a quick visual skimming through—it is imperative to trace the entire paths starting from the output nodes back to the first input in order.

Typed decision graphs (TDGs) [12] offer reduction of BDDs into a graph with a lot fewer nodes and connections. The advantages and drawbacks of the method are therefore mostly similar to those of BDDs, including dependence of the size/complexity of the representation on the ordering of the variables. Also closely related to BDDs are graphical techniques called “implicit graph based method” [13–15] and signal transition graphs (STGs) [16].

The Quine–McCluskey algorithm [17–19] (or method of prime implicants) is functionally identical to Karnaugh mapping and is a deterministic way to check that the minimal form of a Boolean function has been reached. The tabular form makes it more efficient for use in computer algorithms, but, not being graphical, it is not as simple or as intuitive as Karnaugh-maps for use by the designer himself/herself. Petrick’s method (the branch-and-bound method) [20] is a technique for determining all minimum sum-of-products solutions from a prime implicant chart and could be looked upon as an extension to the Quine–McCluskey algorithm. The process gets tedious for a human for a large number of inputs and does not scale well on computers likewise. The Espresso algorithm [21] uses a radically different approach by applying heuristics, manipulating the “cubes” representing the product terms in the ON-, DC-, and OFF-covers iteratively, instead of expanding the minterms. Recently, a new paradigm has been proposed called “Majority-Inverter Graph” [22], which is a directed acyclic graph consisting of three-input majority nodes and direct/negated edges.

Keeping classroom education of Boolean functions at the focus, several methodologies [23–27] that are mostly nongraphical and can best be demonstrated with systems with only a limited number of inputs have been proposed, in addition to more generalised software implementations [28–32] not directly useful for demonstrating the concepts through easy hand-calculations. Karnaugh-maps—due to the method’s graphical nature—continue to be used as the first method to explain the optimization process in the switching theory textbooks [33] still today.

In the context of the comparison of the methods noted so far, we propose that the following are the desired characteristics of an ideal logic representation methodology:

- (i) It is not constrained by the number of inputs or outputs and can in theory handle an infinite number of inputs and outputs. That is to say, multiple output functions can be represented simultaneously.
- (ii) The optimization process should be intuitive; ideally visual inspection should be enough to establish input-output relationship. Or it should be based on a paradigm that is intuitive enough for a human to simply hand-calculate. Graphical methods like K-maps do offer that advantage.
- (iii) If it is graphical, an equivalent numerical method should exist, which can effectively be used as a data structure to optimally represent combinational logic through conventional programming.
- (iv) Translating a Boolean equation into the graphical representation (or vice versa) is a simple process.
- (v) Ordering of variables should play no role in the complexity of the representation.

The proposed technique satisfies all of the desired characteristics listed, except the very last one. Ordering of variables contributes to the ease with which a human can optimize a function without using any computing resources.

2. Definitions

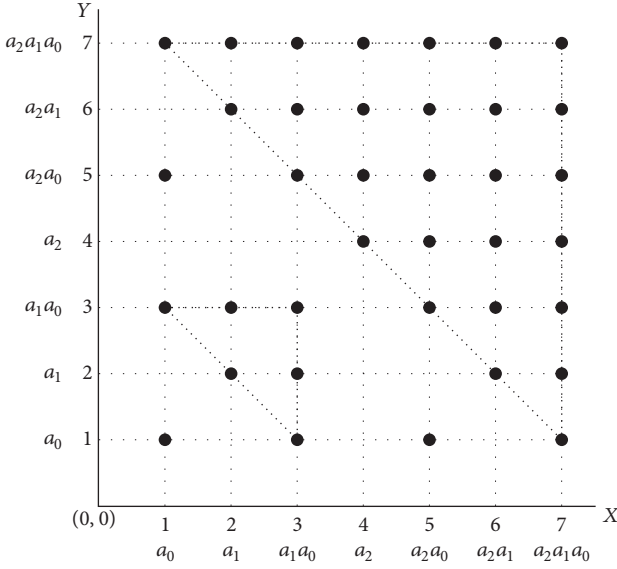
- (i) The logical operators/symbols “ \neg ”, “ $+$ ”, “ \cdot ”, “ \oplus ”, and “ \odot ” denote the logical negation, OR, AND, XOR, and XNOR operations, respectively.
- (ii) The symbols “ \prod ” and “ \sum ” denote the logical AND and OR operations of the variables in iteration, respectively.
- (iii) $\{a_0, a_1, a_2, \dots, a_i, \dots, a_n\}$ is a set of inputs to the combinational logic block under consideration.
- (iv) ϕ denotes an empty set.
- (v) SOP = sum of products.
- (vi) POS = product of sums.

3. Proposed Technique (Pandit-Plot)

Every Boolean equation is represented by a scatter plot in two dimensions. In effect, in its simplest version, corresponding to four quadrants, four Boolean expressions can simultaneously be represented and optimized on a single plane. By using different markers, one for each function, multiple logic functions may be represented in a single plane and subsequently reduced.

3.1. Coordinate System and Input Correspondence. Every integer coordinate “ p ” corresponds to an input combination $a_{i_1}, a_{i_2}, \dots, a_{i_m}$ such that

$$|p| = \sum_{k=1}^m 2^{i_k} \quad (p \in \mathbb{Z}, i_k \in \mathbb{N}). \quad (1)$$

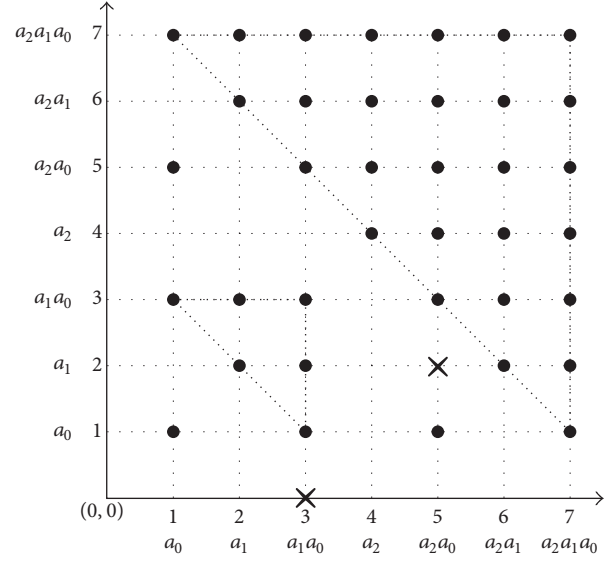
FIGURE 1: Pandit-Plot (for $n = 3$) with redundant data-points.

One axis (say x -axis) represents a condition where the designated inputs are all True (T) (also called High (H) or logic level 1), while the other represents a case where those inputs are all False (F) (also called Low (L) or logic level 0). There is, therefore, a one-to-one correspondence between the set of all of combinations of the inputs and the set of points on the plane with integer coordinates. With such a representation system in place, every point on the plot with integer coordinates signifies a single product (in a SOP representation) or a single sum (in a POS representation). As an example, for a system with three inputs (a_0 , a_1 , and a_2), the template of the plot that is restricted to the first quadrant ($p \geq 0$) will be as shown in Figure 1.

Throughout the discussion henceforth, we only consider the template in the first quadrant. The templates in four quadrants are symmetric to each other, with x - and y -axes and the “ $x = y$ ” and “ $x = -y$ ” lines as the axes of symmetry. We observe that, in the scatter plot from Figure 1, certain data-points have been marked distinctively using the filled circles. These will be called “redundant” input combinations—for the reasons that will be explained in Section 3.3.

3.2. Representing an Input Combination. Representation of the combinational function “ f ” in graphical form may be done for “ f ” either in POS or in SOP form; the choice lies with the logic designer/user of the plot.

To represent a function in SOP form, every constituent product term is separated into two products, one containing exclusively the direct inputs and the other only the negated inputs. The overall product is represented by a point (x, y) , where, for example, the x -coordinate corresponds to the set of input combinations coming only from the direct inputs, while the y -coordinate corresponds to the set of input combinations from only the negated inputs, using (1). The absence of inputs implies that the corresponding coordinate is 0.

FIGURE 2: Pandit-Plot for $f = a_0a_1 + a_0\neg a_1a_2$.

As an example, let us consider the following sum of products:

$$f = a_0a_1 + a_0\neg a_1a_2, \quad (2)$$

represented by points (3, 0) and (5, 2), shown in Figure 2.

- (i) For the first product “ a_0a_1 ” to be True, both a_0 and a_1 need to be True. Therefore, the x -coordinate $= 2^0 + 2^1 = 3$, as per (1). Because there are no negated inputs in the first product, no a_k needs to be False for f to be True. Therefore, the y -coordinate is 0. The product “ a_0a_1 ” is, therefore, represented by point (3, 0).
- (ii) Similarly, for the product “ $a_0\neg a_1a_2$ ” to be True, both a_0 and a_2 need to be True. Therefore, the x -coordinate $= 2^0 + 2^2 = 5$, as per (1). Also, a_1 needs to be False for function f to be True. Therefore, the y -coordinate $= 2^1 = 2$. The product “ $a_0\neg a_1a_2$ ” is, therefore, represented by point (5, 2).

Points (3, 0) and (5, 2) will be called “data-points” as they summarise the function by transforming it into numerical data, corresponding to the chosen coordinate system.

Similarly, for a POS-based representation, we look for False (0/F/L) logic outputs to mark the points transforming every sum in the expression.

Every Boolean function can therefore be represented by only two parameters alone in this way, namely, the coordinates of the representative points, irrespective of the number of inputs. This is a huge data compression in itself, where instead of storing the entire truth table, or even all of the combinations for which the output is either T or F (whichever count is less), we can simply represent a function with a few data-points. The system can further be optimized to remove extraneous information as will be explained in Sections 3.3, 3.5, and 3.6.

```

(1) procedure GENERATE1( $n$ )           ▷  $n$  = number of inputs
(2)    $RedundantSet = \phi$ 
(3)   for  $x \leftarrow 0, 2^n - 1$  do       ▷ Represents direct inputs
(4)     for  $y \leftarrow 0, 2^n - 1$  do ▷ Represents negated inputs
(5)        $x_{binary} \leftarrow (x)_2$ 
(6)        $y_{binary} \leftarrow (y)_2$ 
(7)        $D \leftarrow elementwise(x_{binary} \wedge y_{binary})$ 
(8)       if  $any(D) = 1$  then
(9)          $RedundantSet = RedundantSet \cup (x, y)$ 
(10)      end if
(11)    end for
(12)  end for
(13) end procedure

```

ALGORITHM 1: Redundant set generation for Pandit-Plot.

3.3. Existence of Redundant Data-Points. It can easily be observed that there exist points on the plot with integer coordinates that correspond to a Boolean product or a Boolean sum that can never occur in a reduced SOP or a reduced POS expression, respectively.

As an example, point (1,1) corresponds to the $(a_0 \neg a_0)$ product in SOP which evaluates to 0/F, and the same point corresponds to the $(a_0 + \neg a_0)$ sum in POS which evaluates to 1/T. These input combinations do not have any effect on the final expression evaluation, as the output corresponding to these input combinations is preknown and therefore redundant to be included as part of any Boolean expression (e.g., $f + a_0 \neg a_0 = f$). We therefore discard such input cases and term these as “redundant points” in the discussions below, while the rest will be called “valid” points, except the origin, which falls into neither of the two categories, representing simply the trivial case of the output being independent of the inputs.

3.4. Generation of the Template. To translate any logic function to this plot, the first step is to build the necessary template or the canvas with all the input redundancies indicated (Figure 1). Such a template may be generated by exploiting patterns for which the redundancies appear (Algorithm 1).

For example, we note the following for a system with n inputs:

- (i) The x - and y -axes extend from 0 to $2^n - 1$.
- (ii) The points on the line $x = y$ are redundant points.
- (iii) All of the upper diagonal elements are redundant points.
- (iv) Redundant points because of the a_0 input’s logic level extend along both the x and y directions skipping every other point, that is, skipping the odd integer coordinates. The redundant points associated with a_0 input are therefore of the form $(2i + 1, 2j + 1)$, where i, j are integers.
- (v) Alternatively, the template may be generated using cellular automata rules on a brick-wall-like automata template as devised in Figure 3 (in contrast to the

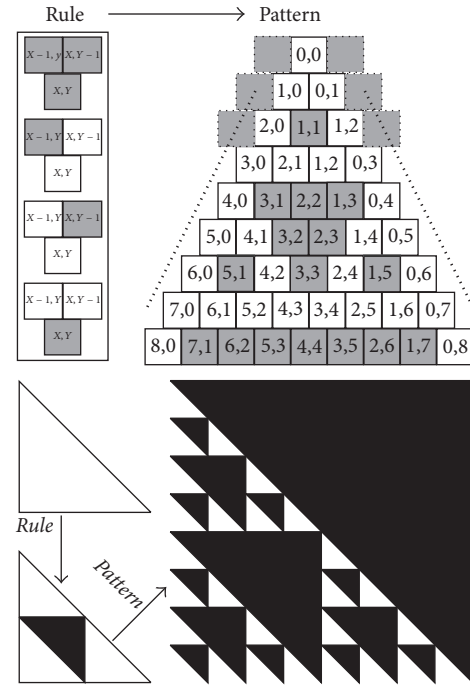


FIGURE 3: Template generation from 2 different kinds of rules, signifying its relation to the Sierpinski triangle/fractals and the cellular automata.

typical cellular automata template as used in Conway’s game of life [34, 35]—where the borders of the individual cells coincide). This gives rise to the Sierpinski triangle [36, 37] pattern, as desired. The captions for the individual blocks indicate the (x, y) coordinates of the points that the blocks represent.

3.5. Logic Laws. The logic optimization process is visual, intuitive, and simple—just as for Karnaugh-maps. It may alternatively be easily implemented using a computer program by understanding the strategies in place numerically. The basic idea of optimization is to get rid of excess of

information by inspecting the scatter plot and bring the data-points as close to the axes as possible. There exist patterns that need to be utilised to effectively carry out this “reduction.” These patterns need to be formulated into a compilation of “logic laws.” A few examples of a logic law or an optimization strategy follow.

Theorem 1 ($\sqrt{2}$ -integer-multiple hypotenuse rule). *If two of the data-points are vertices of an upright right angled isosceles triangle with its equal sides having length $= 2^k$ ($k \in \mathbb{N}$) and if the third vertex is not to the right of any of those two data-points, then the two data-points could effectively be represented by that third vertex alone, so long as the third vertex is not a redundant point.*

Proof. Let data-points be $A(x_1, y_1)$ and $B(x_2, y_2)$, such that $x_1 > x_2$ and $y_1 < y_2$ and $(x_1 - x_2) = (y_2 - y_1) = 2^k$, $k \in \mathbb{N}$, without loss of generalisation.

Because none of the three points, $A(x_1, y_1)$, $B(x_2, y_2)$, and $C(x_2, y_1)$, is a redundant point and because $x_1 + 2^k = x_2$, it follows that

$$x_1 = \sum_{i=\text{ind}(x_1)} 2^i = 2^k + \sum_{i=\text{ind}(x_2)} 2^i, \quad (3)$$

where $\text{ind}(p)$ is set of suffixes corresponding to inputs associated with coordinate “ p .”

Similarly, it follows that

$$y_2 = \sum_{i=\text{ind}(y_2)} 2^i = 2^k + \sum_{i=\text{ind}(y_1)} 2^i. \quad (4)$$

Therefore, the point $B(x_2, y_2)$ represents both the direct and negated inputs corresponding to the point $C(x_1, y_2)$, along with the input $\neg a_k$. Similarly, the point $A(x_1, y_1)$ represents both the direct and negated inputs corresponding to the point $C(x_1, y_2)$, along with the input a_k . The two points, A and B , therefore, effectively represent the inputs corresponding to the point C and, in addition, either the direct or the negated version of the input a_k .

Therefore, in SOP form,

$$f := f_A + f_B = (a_k) f_C + (\neg a_k) f_C = f_C, \quad (5)$$

and, in POS form,

$$f := f_A f_B = (a_k + f_C) (\neg a_k + f_C) = f_C, \quad (6)$$

where $f_G :=$ function associated with point G .

Thus, as per (5) and (6), this rule is simply a manifestation of the two well-known equalities (1) $P(q) + P(\neg q) = P$ and (2) $(S + q)(S + \neg q) = S$ for the template we present here—where P and S are, respectively, the product terms and the summation terms consisting of variables excluding q . \square

As an illustration, if the plot in Figure 4 represents a Boolean function f in SOP form with the points A and B as its “data-points,” then per the definitions earlier

$$f = a_0 \neg a_1 a_2 + a_0 \neg a_1 \neg a_2. \quad (7)$$

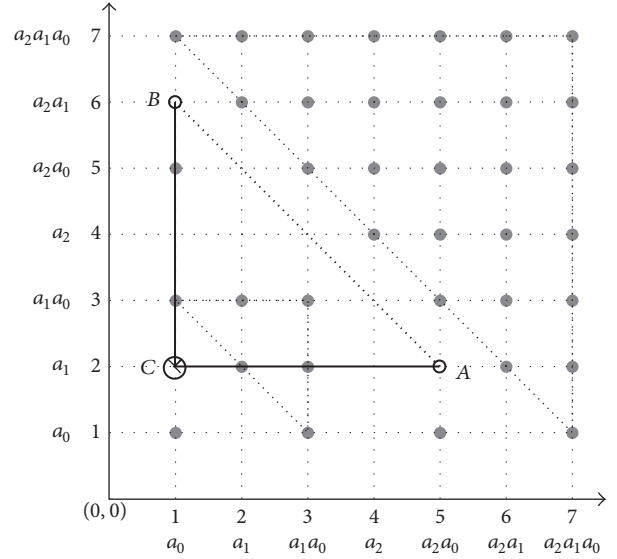


FIGURE 4: Diagonal optimization for SOP.

This is because point $A(5, 2)$ translates to “ $a_0 \neg a_1 a_2$,” while point $B(1, 6)$ represents “ $a_0 \neg a_1 \neg a_2$.”

Per the law proposed, points A and B are represented together by point $C(1, 2)$ alone, since the right angled triangle ABC is an isosceles triangle with its congruent sides having length 2^k , $k \in \mathbb{N}$ ($l(CA) = l(CB) = 4$).

We find here that the hypothesis is true, since

$$\begin{aligned} f &= a_0 \neg a_1 a_2 + a_0 \neg a_1 \neg a_2 = (a_0 \neg a_1) (a_2 + \neg a_2) \\ &= a_0 \neg a_1 \implies C(1, 2). \end{aligned} \quad (8)$$

If the scatter plot in Figure 4 were to correspond to a POS representation, then the plot would have been represented as

$$f = (a_0 + \neg a_1 + a_2) (a_0 + \neg a_1 + \neg a_2). \quad (9)$$

Effectively,

$$\begin{aligned} f &= (a_0 + \neg a_1) + (a_0 + \neg a_1) (a_2 + \neg a_2) + (a_2) (\neg a_2) \\ &= (a_0 + \neg a_1) \implies C(1, 2). \end{aligned} \quad (10)$$

In either case, points $A(5, 2)$ and $B(1, 6)$ are together represented by $C(1, 2)$ alone.

Theorem 2 (symmetry about “ $x = y$ ” line). *Two “valid” points symmetric about the “ $x = y$ ” line in an SOP plot can be effectively represented by a newly defined logical operation represented by operator \star , where*

$$\begin{aligned} \{a_i\} \star \{a_j\} &= (\prod a_i) \cdot (\prod \neg a_j) + (\prod \neg a_i) \cdot (\prod a_j) \\ \text{such that } \{a_i\} \cap \{a_j\} &= \phi. \end{aligned} \quad (11)$$

Two “valid” points symmetric about the “ $x = y$ ” line in a POS plot can be effectively represented by a newly defined logical operation represented by operator $\tilde{*}$, where

$$\{a_i\} \tilde{*} \{a_j\} = (\sum a_i + \sum \neg a_j) \cdot (\sum \neg a_i + \sum a_j) \quad (12)$$

such that $\{a_i\} \cap \{a_j\} = \phi$.

Proof. Two points symmetric about the “ $x = y$ ” line will be of the forms $A(x_0, y_0)$ and $B(y_0, x_0)$. It follows that the set of direct inputs corresponding to the point A ($:= \{a_i\}$) is the same as the set of negated inputs that are represented by point B and vice versa ($:= \{a_j\}$).

Because points A and B are not redundant points, it follows that $\{a_i\} \cap \{a_j\} = \phi$.

Therefore, for SOP,

$$\begin{aligned} \therefore f_A &= (\prod a_i) \cdot (\prod \neg a_j), \\ \therefore f_B &= (\prod \neg a_i) \cdot (\prod a_j), \end{aligned}$$

such that $\{a_i\} \cap \{a_j\} = \phi$, (13)

\Downarrow

$$\begin{aligned} f &= f_A + f_B \\ &= (\prod a_i) \cdot (\prod \neg a_j) + (\prod \neg a_i) \cdot (\prod a_j). \end{aligned}$$

Similarly, for POS,

$$\begin{aligned} \therefore f_A &= (\sum a_i + \sum \neg a_j), \\ \therefore f_B &= (\sum \neg a_i + \sum a_j), \end{aligned}$$

such that $\{a_i\} \cap \{a_j\} = \phi$, (14)

\Downarrow

$$f = f_A \cdot f_B = (\sum a_i + \sum \neg a_j) \cdot (\sum \neg a_i + \sum a_j).$$

□

For a system with just two inputs, the “ $*$ ” operator represents simply an “XOR” (\oplus) operation, while the “ $\tilde{*}$ ” operator represents an “XNOR” (\odot) operation. The “NAND-only” implementation of “ $*$ / $\tilde{*}$ ” operation is shown in Figure 5. This newly defined logic gate can then be optimized for given fan-in and fan-out values as necessary to be used as a building block in the subsequent logic design.

3.6. Process of Optimization. While there is much scope for further research in this area for effective algorithm development in terms of logic optimization, including the ones targeted to novel topologies such as the ones involving IMPLY ($\neg A + B$) gates alone, for example (instead of SOP/POS forms presented here, effectively implemented using “NAND alone” and “NOR alone” topologies, resp.), one of the most likely logic flows for such optimization is presented next.

Steps involved in this example algorithm are as follows:

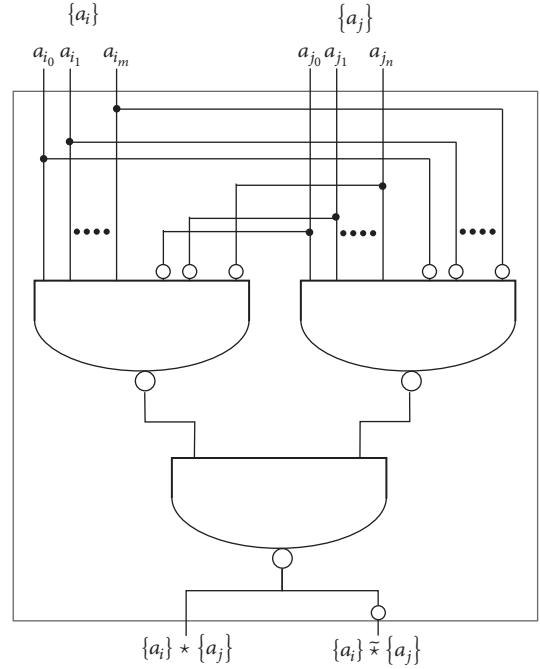


FIGURE 5: “NAND-only” implementation of the newly defined logical operations “ $*$ and $\tilde{*}$ ” that are essentially \oplus (XOR) and \odot (XNOR) operations, respectively, for a two-input system.

- (i) “Generate” the template and plot the function.
- (ii) “Expand” the expression to include every possible product in SOP form (every possible sum for a POS form).
- (iii) “Reduce” the expression using logic laws.
- (iv) “Repeat” the above two steps in iteration until no further reduction is possible, as illustrated in Figure 6.

As an example, let $f = (\neg a_1 + a_0 a_1 a_2)$, and let the representation form chosen be SOP-based as well. f ultimately gets reduced to $f = \neg a_1 + a_0 a_2$ as shown in Figure 6.

- (1) Generate: plot the product terms (Figure 6(a)).
- (2) Expand: plot points relevant to each of the current data-points (Figure 6(b)). Here, $\neg a_1$ is expanded to include points wherever $\neg a_1 = 1 \implies f = 1$.
- (3) Reduce: reduce the expression using logic laws. If a new data-point, replacing one or some of the original data-points, is obtained through this process and if this data point cannot be further reduced, then it is noted and kept for the next iteration. (We note an example of such data-point with a label “NEW” in Figure 6(c).)
- (4) Repeat: repeat steps (2) and (3) until no new replacement data-point closer to the origin can be obtained (Figure 6(d)).

Similar to the Karnaugh-maps, the “do not care” input combinations could be exploited for logic reduction [3, 38].

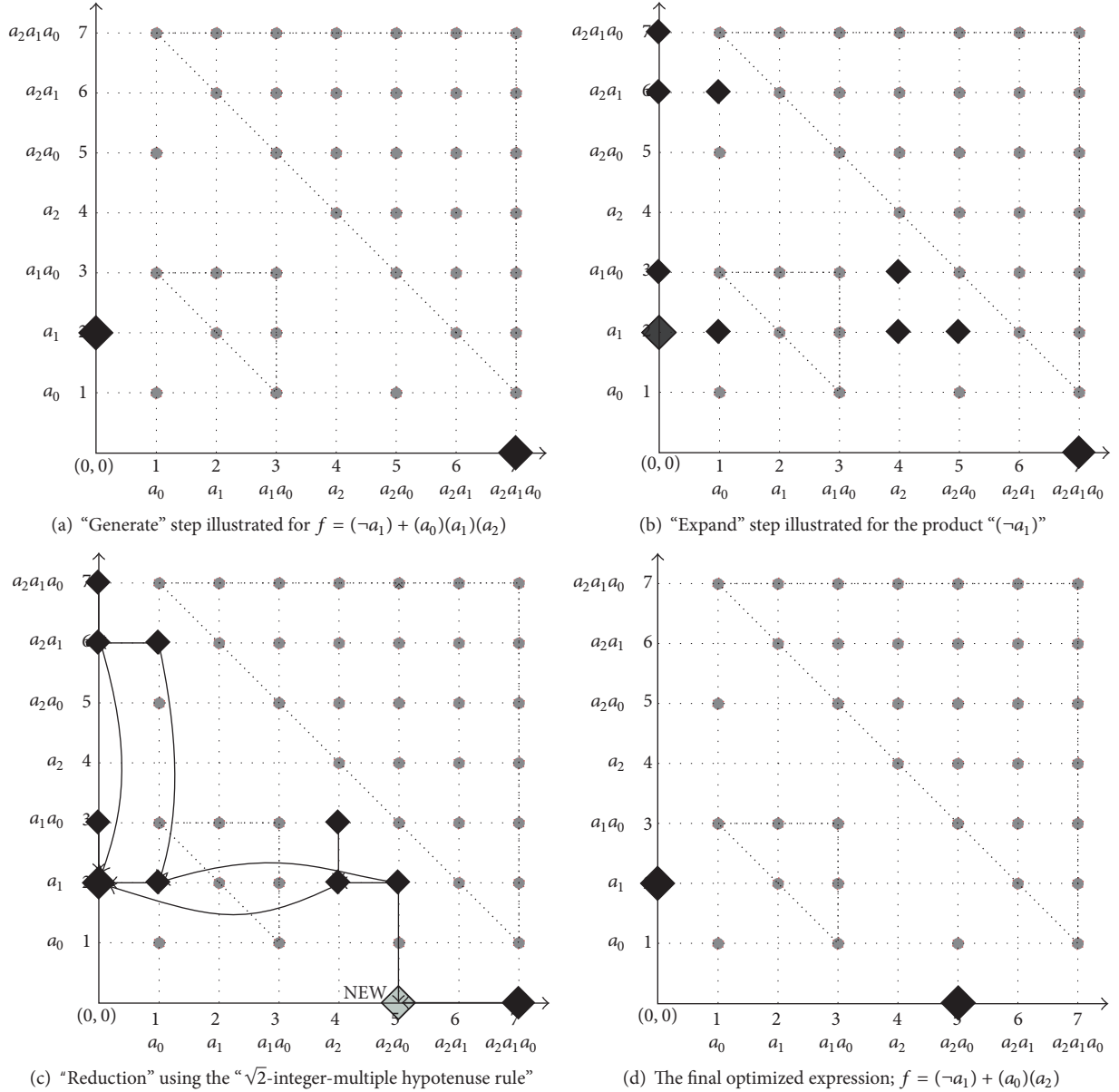


FIGURE 6: Process of optimization; the steps in order are illustrated.

3.7. As a Data Structure. Because the individual Boolean terms are represented by only two coordinates (irrespective of the number of inputs), any combinational logic may be represented by two parameters alone. This can be viewed as a conversion of a graphical scheme to an equivalent nongraphical numerical scheme, to be handled by computing systems efficiently. The two coordinates may be stored in the memory as a single unit.

Because one can represent any logic expression through a set of coordinates alone, we now investigate the number of bits required to store each of the numbered coordinates. While a proposed template for n inputs consists of 4^n points (including the origin), the number of valid coordinates is only

$3^n - 1$. This is because a variable can manifest itself in an input combination in its original form (a_k) or as a negated version ($\neg a_k$) or it could simply be absent. Thus, there are three possibilities for every input a_k . The number of valid points is, therefore, $3^n - 1$ (excluding the origin indicative of a trivial case where the output is independent of all of the inputs under consideration). Both the total number of points ($\tau = 4^n - 1$) in a template and the number of valid points ($v = 3^n - 1$) grow exponentially with n . The ratio of the number of bits thus required (with and without considering the redundant points) = $\text{ceil}(\log_2(3^n - 1)) / \text{ceil}(\log_2(4^n - 1))$ which varies between 0.75 and 0.85 and eventually converges to 0.8 for large n . As an example, for $n = 3$, $v = 26$, and

$\tau = 63$, while, for $n = 6$, $v = 728$, and $\tau = 4095$. Memory savings, thus, grow with n if we effectively take into account the redundancies in the template.

However, in comparison to K-maps ($= 2^n$ cells for an n -input system), our proposed method requires a much higher number of points ($= 3^n - 1$, if we consider only the valid input combinations, excluding the origin). This is because the K-maps do not take into account all of the input combinations, but only the ones where each of the inputs manifests itself by its presence in its original or in its negated form (thus, 2^n possibilities). In the proposed method, we explicitly take into account also the reduced input combinations where some of the inputs are absent.

3.8. Possible Extensions and Further Research Scope

- (i) Beyond SOP/POS paradigms: more patterns and the resulting optimization strategies need to be investigated in different logic gate methodologies, for example, SOP (“AND-OR”, which is “NAND-only”), POS (“OR-AND”, which is “NOR-only”), and “IMPLY-only” realisations.
- (ii) Multilevel logic: the concept could be extended to three dimensions to represent a ternary logic system, where both inputs and outputs can have three possible logic levels (0, 1, 2). The resulting representation is, thus, in the form of a “logic lattice” instead of the two-dimensional grid presented here, where the third dimension would serve a dual purpose. First, it represents a condition where all the inputs are logical level “2” (just as Z and Y represented states “1” and “0,” resp.). Secondly, the z -axis may also represent an output logic level within such a “logic slice.” As an example, output level-1 may correspond to a point with integer (X, Y, Z) coordinates, level-0 may correspond to a point just beneath this point, and level-2 just above (Figure 7).
- (iii) Alternate coordinate systems: every data-point symbolises a huge degree of information, but with only two or three parameters (coordinates). Because this information need not necessarily come about using a Cartesian coordinate system alone, one can choose to alternatively employ some other coordinate system and look for patterns and laws for optimization within such schemes. For example, a 2D polar system may be used, where the (x, y) to (r, θ) transformation is governed by the equations $r = x$, $\theta = 2\pi y/(2^n - 1)$, and $x, y \in [0, 2^n - 1]$. Note that θ remains in the range $[0, 2\pi]$ irrespective of the number of inputs (Figure 8). The logic laws for such alternate coordinate systems—where every coordinate of interest ψ follows the equation $\psi = f(\sum_{k=1}^m 2^{i_k})$, $i_k \in \mathbb{N}$ —can be devised by recognising the inherent patterns, which remains an open problem with further scope for research.

3.9. Implications to Other Fields of Study. One can use the template to establish three algebraic equalities, as will be discussed in detail in this section. The method, in general,

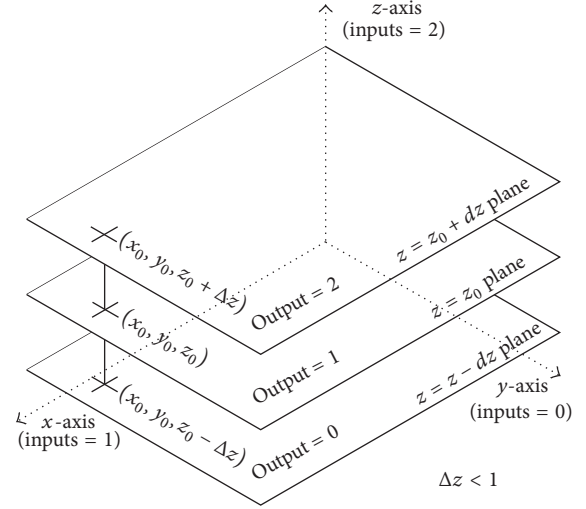


FIGURE 7: Ternary logic representation ($x_0, y_0, z_0 \in \mathbb{N}$). The point $(x_0, y_0, z_0 \pm \Delta z$ or $z_0)$ represents the input condition, where all inputs corresponding to x -coordinate = 1, all inputs corresponding to y -coordinate = 0, and all inputs corresponding to z -coordinate = 2.

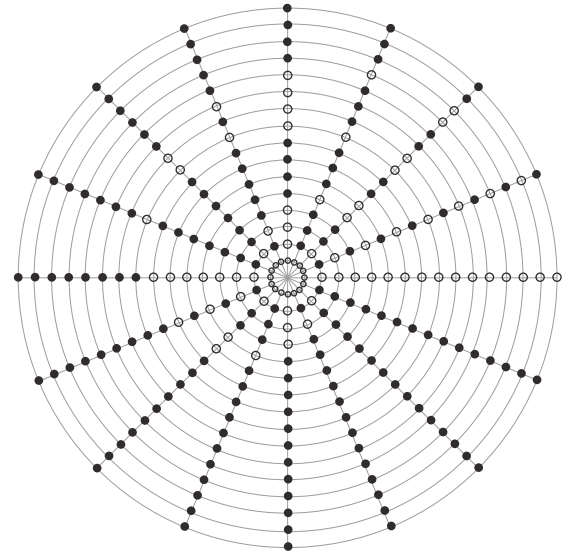


FIGURE 8: Cartesian (x, y) coordinate system converted into polar (r, θ) system, where $r = x$, $\theta = 2\pi y/(2^n - 1)$, and $x, y \in [0, 2^n - 1]$. Filled circles indicate the “redundant points.”

is useful for alternative representation and processing of data whenever there is representation of information (i.e., encoding or decoding) in the form of distinct or discrete states. The basic representation strategy can therefore be extended to represent and optimize operations on qubits [39] or to represent codon tables [40, 41] or design digital synthetic gene circuits [42]. Using the template used in the method proposed, we have derived the following three equalities in relation to $(4^n - 3^n)$. The method gives practical evidence of the equalities invented and presented next.

Theorem 3.

$$3^n + \left[\sum_{i=0}^{n-2} 3^{n-2-i} 2^i (2^{i+1} - 1) \right] + (2^n - 1) 2^{n-1} = 4^n, \quad (15)$$

$$\forall n \in \mathbb{N}, n \geq 2,$$

or, in other words,

$$3^n + \left[\sum_{i=0}^{n-2} 3^{n-2-i} \binom{2^{i+1}}{2} \right] + \binom{2^n}{2} = 4^n, \quad (16)$$

$$\forall n \in \mathbb{N}, n \geq 2.$$

Theorem 4.

$$3^n + \left[\sum_{i=0}^{n-1} 3^i 4^{(n-1-i)} \right] = 4^n, \quad \forall n \in \mathbb{N}, n \geq 2. \quad (17)$$

Theorem 5. As a consequence,

$$\left[\sum_{i=0}^{n-1} 3^i 4^{(n-1-i)} \right] = \left[\sum_{i=0}^{n-2} 3^{n-2-i} \binom{2^{i+1}}{2} \right] + \binom{2^n}{2} \quad (18)$$

$$\forall n \in \mathbb{N}, n \geq 2$$

\because Equations (16) and (17).

Let us first establish that the equalities being discussed are true, using induction.

Proof. For Theorem 3, we observe that the equalities presented in (15) or (16) are true for $n = 2$ and $n = 3$.

For $n = 2$,

$$\begin{aligned} \text{L.H.S.} &= 3^2 + \left[\sum_{i=0}^0 3^{0-i} (2^{i+1} - 1) 2^i \right] + (2^2 - 1) 2^1 \\ &= 9 + [(1(2-1)1)] + (4-1)2 = 9 + 1 + 6 \\ &= 16 = 4^2 = \text{R.H.S.} \end{aligned} \quad (19)$$

For $n = 3$,

$$\begin{aligned} \text{L.H.S.} &= 3^3 + \left[\sum_{i=0}^1 (3^{1-i} (2^{i+1} - 1) 2^i) \right] + (2^3 - 1) 2^2 \\ &= 27 + [(3^1 (2^1 - 1) 2^0) + (3^0 (2^2 - 1) 2^1)] \\ &\quad + (7) 4 = 27 + [3(1)(1) + 1(3)(2)] + (7) 4 \\ &= 27 + 9 + 28 = 64 = 4^3 = \text{R.H.S.} \end{aligned} \quad (20)$$

Now, say, for some $n = m \geq 2$,

$$\begin{aligned} 3^m + \left[\sum_{i=0}^{m-2} (3^{m-2-i} (2^{i+1} - 1) 2^i) \right] + (2^m - 1) 2^{m-1} \\ = 2^{2m} = 4^m. \end{aligned} \quad (21)$$

Therefore,

$$\begin{aligned} 3^m + (3^{m-2} (2^1 - 1) 2^0 + 3^{m-3} (2^2 - 1) 2^1 + \dots \\ + 3^1 (2^{m-2} - 1) 2^{m-3} + 3^0 (2^{m-1} - 1) 2^{m-2}) \\ + (2^m - 1) 2^{m-1} \\ = 2^{2m}. \end{aligned} \quad (22)$$

For $n = m + 1 \geq 2$, therefore,

$$\begin{aligned} \text{L.H.S.} &= 3^{m+1} + \left[\sum_{i=0}^{m-1} (3^{m-1-i} (2^{i+1} - 1) 2^i) \right] \\ &\quad + (2^{m+1} - 1) 2^m \\ &= 3^{m+1} + (3^{m-1} (2^1 - 1) 2^0 + 3^{m-2} (2^2 - 1) 2^1 + \dots \\ &\quad + 3^1 (2^{m-1} - 1) 2^{m-2} + 3^0 (2^m - 1) 2^{m-1}) \\ &\quad + (2^{m+1} - 1) 2^m \\ &= 3(3^m + (3^{m-2} (2^1 - 1) 2^0 + 3^{m-1} (2^2 - 1) 2^1 + \dots \\ &\quad + 3^0 (2^{m-1} - 1) 2^{m-2})) + (2^m - 1) 2^{m-1} \\ &\quad + (2^{m+1} - 1) 2^m \\ &= (2 + 1) \times (2^{2m} - (2^m - 1) 2^{m-1}) + (2^m - 1) 2^{m-1} \\ &\quad + (2^{m+1} - 1) 2^m \quad (\because \text{Equation (22)}) \\ &= (2^{2m+1} - 2^{2m} + 2^m) + (2^{2m} - 2^{2m-1} + 2^{m-1}) \\ &\quad + (2^{2m-1} - 2^{m-1} + 2^{2m+1} - 2^m) \\ &= 2^{2m+2} = 4^{m+1} = \text{R.H.S.} \end{aligned} \quad (23)$$

Therefore, if the statement is true for $n = m$, then the statement is true for $n = m + 1$ ($n, m \in \mathbb{N}$). Because the statement is true for $n = 2$ and 3 , it is true for every $n \in \mathbb{N}$ and $n \geq 2$. \square

Proof. For Theorem 4 similarly, we observe that the equality presented in (17) is true for $n = 2$:

$$\begin{aligned} \text{L.H.S.} &= 3^2 + \left[\sum_{i=0}^1 3^i 4^{(2-1-i)} \right] = 3^2 + [3^0 4^1 + 3^1 4^0] \\ &= 16 = 4^2 = \text{R.H.S.} \end{aligned} \quad (24)$$

Now, say, for some $n = m \geq 2$,

$$3^m + \left[\sum_{i=0}^{m-1} 3^i 4^{(m-1-i)} \right] = 4^m. \quad (25)$$

For $n = m + 1 \geq 2$, therefore,

$$\begin{aligned}
 \text{L.H.S.} &= 3^{m+1} + \left[\sum_{i=0}^m 3^i 4^{(m-i)} \right] \\
 &= 3^{m+1} + 4 \times \left[\sum_{i=0}^{m-1} 3^i 4^{(m-1-i)} \right] + 3^m 4^0 \\
 &= 3^{m+1} + 4 \times [4^m - 3^m] + 3^m 4^0 \\
 &= 3^{m+1} + 4^{m+1} - (3+1) \times 3^m + 3^m = 4^{m+1} \\
 &= \text{R.H.S.}
 \end{aligned} \tag{26}$$

Because the statement is true for $n = 2$, it is true for every $n \in \mathbb{N}$ and $n \geq 2$. \square

In the context of the template for the method proposed, the two theorems can be proven graphically by simply counting the number of valid and redundant points, via different formulations. If $3^n + f(n) = 4^n$ for all $n \in \mathbb{N}$, we know the following:

- (i) $4^n = 2^n 2^n = (\text{axis range})^2 = \text{total number of points}$.
- (ii) $3^n = \text{total number of "valid" data-points, including the origin}$.
 $(\because \text{Every input } a_i \text{ can manifest itself in the input combination represented by } (X, Y), \text{ either in its original form (input combination represented by the } X\text{-coordinate) or as a negated input (input combination represented by the } Y\text{-coordinate) or may not manifest itself at all.})$
- (iii) $f(n) = \text{total number of redundant points}$.
 $(\because \text{Total number of points} = \text{total number of "valid" points including the origin} + \text{total number of "redundant" points}).$

Proof. To establish equality presented in the Theorem 3, we use the formulation of $f(n)$ as presented in Figure 9. From Figure 9, it is evident that

$$\begin{aligned}
 \therefore \sum_{l=1}^k l &= \frac{k(k+1)}{2} \quad \forall k \in \mathbb{N}, \\
 k &= 2^j \quad \forall j \in \mathbb{N}, \quad 0 \leq j < n, \\
 \therefore f(n) &= \left[\sum_{i=0}^{n-2} 3^{n-2-i} 2^i (2^{i+1} - 1) \right] + (2^n - 1) 2^{n-1}.
 \end{aligned} \tag{27}$$

The exponent of 3 translates to a multiplier that represents the frequency of occurrence of the constituent triangle of that specific size in the template. \square

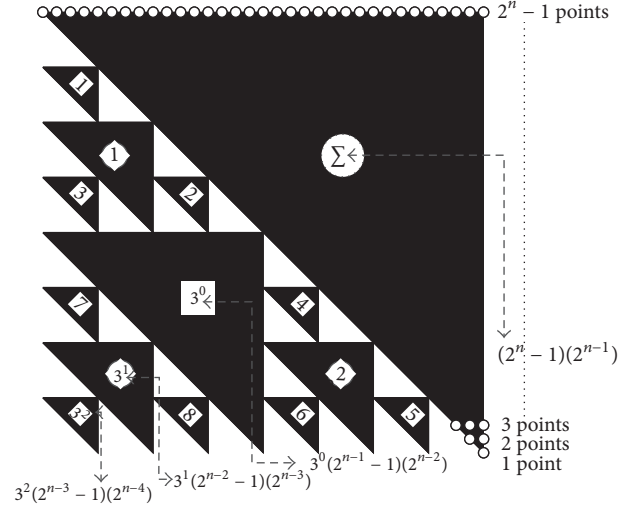


FIGURE 9: Total number of “redundant points”/the points in the darkened region, by counting the number of points in constituent triangles separately.

Proof. Alternately, Theorem 4 can be proven using a recursive formula for $f(n)$, as obtained from Figure 10.

$$\begin{aligned}
 \therefore f(n) &= 3f(n-1) + 4^{n-1} \\
 \therefore f(n) &= 3 \left[3f(n-2) + 4^{n-2} \right] + 4^{n-1} \\
 &= 3^2 f(n-2) + 3 \times 4^{n-2} + 4^{n-1} \\
 &= \left[3^{n-1} f(1) + \sum_{i=0}^{n-2} 3^i (4^{(n-1-i)}) \right] \\
 &= \left[\sum_{i=0}^{n-1} 3^i 4^{(n-1-i)} \right] \quad \because f(1) = 1.
 \end{aligned} \tag{28}$$

\square

This method can be used in classrooms to quickly hand-calculate an optimized form of the logic function using a ruler and a compass alone (Figure 11).

4. Conclusion

A novel graphical technique for digital logic representation and optimization has been proposed, which makes optimization algorithm much more instinctive and easy. Multiple equations can be optimized simultaneously using the template presented. One can, theoretically, therefore optimize simultaneously multiple combinational logic circuits with any number of inputs or outputs. We propose a data structure transformation that can compress a truth table into a few parameters. The approach and the fundamentals involved are likely to generate many novel solutions applicable to other disciplines. A new paradigm is being proposed, which opens up new avenues for research in terms of new methodologies and pattern identification towards better logic reduction

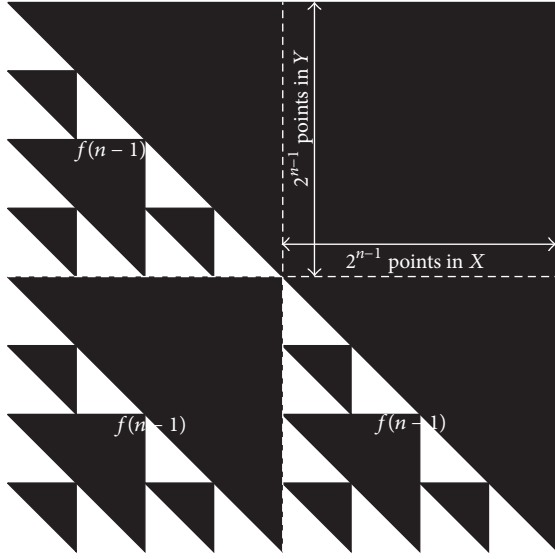


FIGURE 10: Total number of “redundant points”/the points in the darkened region, by deriving a recursive formula for $f(n)$. Thus, $f(n) = 3 \times f(n-1) + (2^{n-1})^2$.

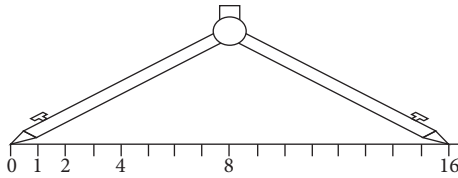


FIGURE 11: A compass and a ruler (marked with 2^n units) can be effectively used to quickly find the points that can be merged together to generate a more optimized logic representation using the logic laws proposed earlier (Section 3.5).

tailored to different implementation topologies such as SOP (“AND-OR”/“NAND-only”), POS (“OR-AND”/“NOR-only”), and “IMPLY-only” realisations or the newer methodologies such as “reconfigurable asynchronous logic automata (RALA)” [43] and the search for most suited template for each in terms of the coordinate system of choice and the template generation rules.

Our proposed technique currently establishes a one-to-one mapping between all of the possible variable combinations and the corresponding output states—representing the absence and presence of every variable (in its original and the negated form) for each of the variable combinations. One possible future research direction is to investigate the relationship between the proposed technique herein and other representation techniques—such as QMDDs which, too, rely on quadrants and one-to-one mappings and which rather extend way beyond the proposed technique to represent transformation matrices featuring complex entries useful for logic involving qubits. Further research may also be targeted towards multilevel discrete systems beyond Boolean algebra and reversible [44, 45] and hazard-free logic synthesis [9].

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

- [1] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, “MIS: A Multiple-Level Logic Optimization System,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, no. 6, pp. 1062–1081, 1987.
- [2] A. J. De Geus and W. Cohen, “A Rule-Based System for Optimizing Combinational Logic,” *IEEE Design Test of Computers*, vol. 2, no. 4, pp. 22–32, 1985.
- [3] M. Karnaugh, “The map method for synthesis of combinational logic circuits,” *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, vol. 72, no. 5, pp. 593–599, 1953.
- [4] M. E. Holder, “A modified karnaugh map technique,” *IEEE Transactions on Education*, vol. 48, no. 1, pp. 206–207, 2005.
- [5] J. Cavanagh, *Computer Arithmetic and Verilog HDL Fundamentals*, CRC Press, 2009.
- [6] Z. Kohavi and N. K. Jha, *Switching and finite automata theory*, Cambridge University Press, 2009.
- [7] C. Y. Lee, “Representation of switching circuits by binary-decision programs,” *Bell Labs Technical Journal*, vol. 38, pp. 985–999, 1959.
- [8] S. B. Akers, “Binary decision diagrams,” *IEEE Transactions on Computers*, vol. 27, no. 6, pp. 509–516, 1978.
- [9] B. Lin and S. Devadas, “Synthesis of Hazard-Free Multilevel Logic Under Multiple-Input Changes from Binary Decision Diagrams,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 8, pp. 974–985, 1995.
- [10] C. Yang and M. Ciesielski, “BDS: A BDD-based logic optimization system,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 7, pp. 866–876, 2002.
- [11] R. E. Bryant, “Graph-based algorithms for boolean function manipulation,” *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.
- [12] L. Mauborgne, “Abstract interpretation using typed decision graphs,” *Science of Computer Programming*, vol. 31, no. 1, pp. 91–112, 1998.
- [13] O. Coudert and J. C. Madre, “A new implicit graph-based prime and essential prime computation technique,” in *Proceedings of the Proc. International Symp. Information Sciences*, Fukuoka, Japan, 1992.
- [14] O. Coudert and J. C. Madre, “A New Graph Based Prime Computation Technique,” in *Logic Synthesis and Optimization*, pp. 33–57, Springer, Boston, MA, USA, 1993.
- [15] O. Coudert, J. C. Madre, H. Fraisse, and H. Touati, “Implicit prime cover computation: An overview,” in *Proceedings of the in Proc. Synthesis and Simulation Meeting and International Interchange (SASIMI)*, Nara, Japan, 1993.
- [16] J. Gu and R. Puri, “Asynchronous circuit synthesis with Boolean satisfiability,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 8, pp. 961–973, 1995.
- [17] W. V. Quine, “The problem of simplifying truth functions,” *The American Mathematical Monthly*, vol. 59, pp. 521–531, 1952.
- [18] W. V. Quine, “A way to simplify truth functions,” *The American Mathematical Monthly*, vol. 62, pp. 627–631, 1955.

- [19] J. McCluskey, "Minimization of Boolean functions," *Bell Labs Technical Journal*, vol. 35, pp. 1417–1444, 1956.
- [20] S. Petrick, "A direct determination of the irredundant forms of a boolean function from the set of prime implicants," Tech. Rep., Air Force Cambridge Research Center, 1956.
- [21] R. L. Rudell, "Multiple-Valued Logic Minimization for PLA Synthesis," Defense Technical Information Center, 1986.
- [22] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Majority-Inverter Graph: A New Paradigm for Logic Optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 806–819, 2016.
- [23] D. C. Fielder, "Classroom Reduction of Boolean Functions," *IEEE Transactions on Education*, vol. 9, no. 4, pp. 202–205, 1966.
- [24] B. C. H. Turton, "Extending Quine-McCluskey for exclusive-or logic synthesis," *IEEE Transactions on Education*, vol. 39, no. 1, pp. 81–85, 1996.
- [25] R. Benzer and V. Rozga, "The Design and Application of a Minority Logic Gate-A Senior Project," *IEEE Transactions on Education*, vol. 10, no. 3, pp. 141–146, 1967.
- [26] N. R. Bell, "A Map Method for the Teaching of the Fundamental Concepts of Compound-Input Logic Circuits," *IEEE Transactions on Education*, vol. 11, no. 3, pp. 173–177, 1968.
- [27] R. F. Tinder, "Multilevel Logic Minimization Using K-map XOR Patterns," *IEEE Transactions on Education*, vol. 38, no. 4, pp. 370–375, 1995.
- [28] B. D. Carroll and I. Chen, "ABAL—A Language for Boolean Function Representation and Manipulation," *IEEE Transactions on Education*, vol. 20, no. 1, pp. 70–72, 1977.
- [29] C. E. Klock, F. R. Schneider, M. V. N. Gomes, D. S. Moura, R. P. Ribas, and A. I. Reis, "KARMA: A didactic tool for two-level logic synthesis," in *Proceedings of the MSE 2007: 2007 IEEE International Conference on Microelectronic Systems Education: Educating Systems Designers for the Global Economy and a Secure World*, pp. 59–60, San Diego, CA, USA, June 2007.
- [30] V. P. Correia and A. I. Reis, "A tutorial tool for switch logic," in *Proceedings of the International Conference on Microelectronic Systems Education, MSE 2001*, pp. 28–29, Las Vegas, NV, USA, June 2001.
- [31] Z. Stanisavljevic, V. Pavlovic, B. Nikolic, and J. Djordjevic, "SDLDS-system for digital logic design and simulation," *IEEE Transactions on Education*, vol. 56, no. 2, pp. 235–245, 2013.
- [32] P. Corsini and L. Rizzo, "SSCSSC: A Tool for the Teaching of Digital Circuits," *IEEE Transactions on Education*, vol. 34, no. 1, pp. 70–75, 1991.
- [33] G. J. Klir and M. A. Marin, "New considerations in teaching switching theory," *IEEE Transactions on Education*, vol. 12, no. 4, pp. 257–261, 1969.
- [34] M. Gardner, "Mathematical games: The fantastic combinations of John Conway's new solitaire game "life"," *Scientific American*, vol. 223, no. 4, pp. 120–123, 1970.
- [35] S. Wolfram, "Cellular automata as models of complexity," *Nature*, vol. 311, no. 5985, pp. 419–424, 1984.
- [36] M. Sierpinski, "Sur une courbe dont tout point est un point de ramification," *Compte Rendus hebdomadaires des séances de l'Académie des Sciences de Paris*, vol. 160, pp. 302–305, 1915.
- [37] J.-P. Allouche and J. Shallit, *Automatic Sequences: Theory, Applications, Generalizations*, Cambridge University Press, 2003.
- [38] K. A. Bartlett, R. K. Brayton, G. D. Hachtel et al., "Multilevel Logic Minimization Using Implicit Don't Cares," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 6, pp. 723–740, 1988.
- [39] P. Niemann, R. Wille, D. M. Miller, M. A. Thornton, and R. Drechsler, "QMDDs: Efficient Quantum Function Representation and Manipulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 86–99, 2016.
- [40] D. Searls and M. Noordewier, "Pattern-matching search of DNA sequences using logic grammars," in *Proceedings of the Seventh IEEE Conference on Artificial Intelligence Application*, pp. 3–9, Miami Beach, FL, USA, 1991.
- [41] C. R. Woese, "Order in the genetic code," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 54, no. 1, pp. 71–75, 1965.
- [42] M. A. Marchisio and J. Stelling, "Automatic design of digital synthetic gene circuits," *PLoS Computational Biology*, vol. 7, no. 2, Article ID e1001083, 2011.
- [43] N. Gershenfeld, D. Dalrymple, K. Chen et al., "Reconfigurable asynchronous logic automata: (RALA)," in *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL10*, vol. 45, pp. 1–6, Madrid, Spain, January 2010.
- [44] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Synthesis of reversible logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 6, pp. 710–722, 2003.
- [45] P. Gupta, A. Agrawal, and N. K. Jha, "An algorithm for synthesis of reversible logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 11, pp. 2317–2329, 2006.

