Multi-output, multi-level, multi-gate design using non-linear programming

A. C. Dimopoulos | C. Pavlatos | G. Papakonstantinou

Summary
The conventional methodology of logic circuits design is by using logic gates. Thus, most of the minimization algorithms concern a limited set of gates (complete sets), like sum of products, exclusive-or sum of products, NAND gates, NOR gates, and so forth. In this paper, a method is proposed for minimizing multi-output Boolean functions using any kind of two-input gates although it can easily be extended to multi-input gates. The method is based on non-linear mixed integer programming. Our methodology was tested against others proposed in the literature showing that it can achieve the same or better results in all cases. The contribution of this work can be summarized in the following: (i) it is the only approach to guarantee minimality to the problem of multi-output, multi-level, multi-gate design, (ii) it is general and flexible and can solve design problems with any predefined gates and gate topology, (iii) it is a novel approach which may be adopted easily to solve other kinds of logic design problems, and (iv) incompletely specified Boolean functions are accepted, in contrary to other methods.

KEYWORDS
Boolean function minimization, incompletely specified functions, logic circuit design, non-linear integer optimization

1 | INTRODUCTION
The conventional method of designing a logic circuit is via using logic gates. Nonetheless, the vast majority of the minimization algorithms involve a limited set of gates (complete sets), for example, sum of products (SOPs), exclusive-or sum of products (ESOPs), NAND gates, NOR gates, and so on. In the literature, there is a very limited number of publications that permit the combination of any kind of gates, such as references. Most of them are using genetic algorithms and no one of them guarantees minimality. However, in previous approaches that guarantee minimality, presented by our team, the method was restricted to only one kind of gates. Here we present a method for minimizing multi-output Boolean functions using any kind of two-input gates, although it can easily be extended to multi-input gates. The method allows the user to select the kind of gates to be used, for example, AND, OR, NAND, NOR, XOR, NOT, and so forth, as well as the architecture of the desired circuit. This architecture can have, for example, the structure of a $p \times q$ grid of gates for at most $q$ functions (multi-output function) or a tree structure for a single-output function.
function. The inputs of each gate can be connected to the outputs of the gates at any previous level of gates in the grid or they can be variables or constants.

The method is based on non-linear mixed integer optimization methods, which minimizes the number of gates used in the structure or the number of transistors within the circuits. For this goal, any suitable optimizer can be utilized. In our case, we choose to execute our benchmark tests on the free access NEOS server \(^{8-11}\) and largely the BARON solver.\(^ {12}\) BARON solver determines the global optima by using branch-and-bound deterministic algorithms. This solver requires fairly general assumptions, which are fulfilled in our case. Therefore, we are able to detect an exact solution, which provides the minimum number of gates. For facilitating the creation of GAMS nomenclature which is required by the NEOS server, a FORTRAN program has been implemented for the automatic translation of the problem description to GAMS. Our methodology was tested against others proposed in the literature showing that it can achieve the same or better results in all cases. However, other methods do not ensure that they produce the minimal solution, while the contributions of the proposed methodology are as follows:

- It is the only approach to guarantee minimality to the problem of multi-output, multi-level, multi-gate design.
- It is general and flexible, since it solves design problems with any predefined gates and gate topology, for example, cascade or triangular for a single-output function and orthogonal mxn for multi-output functions.
- It is a novel approach which may be adopted easily to solve other kinds of logic design problems.

2 | THEORETICAL BACKGROUND

In the following, all basic Boolean algebra definitions necessary for the understanding of this paper are provided.

**Definition 1.** Let us consider \(f\) a function that maps \(f : \{0,1\}^n \rightarrow \{0,1\}\).

**Definition 2.** If \(x\) is a Boolean variable having as domain the set \(V = \{0,1\}\) and \(M \subseteq V\), then \(x^M\) is a literal of \(x\), such that \(x^M = 1\) in case \(x \in M\) and \(x^M = 0\) in case \(x \in V \setminus M\). In case \(M = V\), then \(x^M = 1\).

**Definition 3.** If \(\bar{x}_i\), when \(i \in [1,n]\) is a literal of the variable \(x_i\), then the term \(T = \bar{x}_1\bar{x}_2...\bar{x}_n\) is a product term or cube. When \(\bar{x}_i = x_i\) or \(\bar{x}_i(\text{excluding } \bar{x}_i = 0,1)\), \(T\) is defined as minterm and \(\bar{x}_i\) is denoted as \(\bar{x}_i\) for all \(i\)’s.

In total, the number of minterms is \(2^n\).

**Definition 4.** In case in a minterm all \(x_i\) are replaced by 1 and all \(\bar{x}_i\) by 0, then a binary number \(g\) called representative number is formed, which corresponds to the specific minterm.

**Definition 5.** For every minterm having \(g\) as representative number and for every Boolean variable \(i\) part of the minterm, the representative bit \(b_{g,i} , i \in [1,n]\), can be defined as

- \(b_{g,i} = 1\) when \(x_i\) exists in the minterm
- \(b_{g,i} = 0\) when \(\bar{x}_i\) exists in the minterm

Every Boolean function \(f\) has a unique representation using the notation of sum of all minterms for which \(f(\bar{x}_1,\bar{x}_2,...,\bar{x}_n) = 1\).

**Definition 6.** A minterm is considered to cover a product term when the product term is equal to 1 and the minterm is equal 1.

**Definition 7.** The length of the bit vector representing a Boolean function \(n\) variables is \(2^n\). The \(g^{th}\)bit \((0 \leq g \leq 2^n - 1)\) equals to 1 when the minterm expression includes the minterm with representative number \(g\). In other case, the \(g^{th}\)bit equals to 0.
**Definition 8.** An expression of the form $\bigoplus \sum_{i=1}^{m} C_i$ that represents a function in a non-unique way is defined as an ESOP terms. In the previous expression, $C_i$ are cubes and $\bigoplus$ is the XOR boolean function. In order to uniquely represent a function, all $C_i$ should be minterms.

**Definition 9.** Let us consider the function $g(x)$, where $x$ is the set of its binary variables $x_i$. For each variable $x_i$, subfunctions of $g$ can be defined as $g(x_1, x_2, ..., x_i = 0, ..., x_n), g(x_1, x_2, ..., x_i = 1, ..., x_n)$, $\{g(x_1, x_2, ..., x_i = 0, ..., x_n) \bigoplus g(x_1, x_2, ..., x_i = 1, ..., x_n)\}$. According to this definition, in the following, the subfunctions will be referred as $g^0, g^1$, and $g^2$, respectively.

According to Definition 9, a Boolean function $g$ may be represented as

$$g(x) = x_0g^0 \bigoplus x_1g^1 = x_0g^2 \bigoplus g^1$$

(1)

These representations are well known as Shannon, Positive Davio, and Negative Davio expansions, respectively. Moreover, the Shannon expansion may also be found as

$$g(x) = x_0g^0 + x_1g^1$$

(2)

**Theorem 1.** For each $n$ variable Boolean function, there is a $3^n$ gates (at most) implementation.

# 3 | THE BASIC IDEA

In this section, an overview of the basic idea is given, where the input of each gate can either be binary variables $x_i$ of a Boolean function, or binary variables that are the output of previous level gates, or a constant 0 or 1. Each gate’s type, input, and output are variable that need to be calculated. Hence, a set of relations should be formed for each minterm $g$:

- Each gate type $t$ is expressed by the following expression:

  $t_{sel_i} \times f_1 + t_{sel_i} \times f_2 + t_{sel_i} \times f_3 + t_{sel_i} \times f_4 + t_{sel_i} \times f_5 + t_{sel_i} \times f_6 + t_{sel_i} \times f_7$, where $t_{sel_i}$ are binary unknown variables to select a gate type and the expression sum should be 1. The $f_j$s are defined as

  $f_1 = \text{AND}(x, y) \Rightarrow x \times y$.
  $f_2 = \text{OR}(x, y) \Rightarrow x + y - (x \times y)$.
  $f_3 = \text{NOT}(x) \Rightarrow (1 - x)$.
  $f_4 = \text{XOR}(x, y) \Rightarrow x + y - 2 \times (x \times y)$.
  $f_5 = \text{NAND}(x, y) \Rightarrow (1 - x) + (1 - y) - (1 - x) \times (1 - y) = 1 - x \times y$.
  $f_6 = \text{NOR}(x, y) \Rightarrow (1 - x) \times (1 - y)$.
  $f_7 = \text{CON}(x) \Rightarrow x$.

  It is noted that the above expressions show the relation between Boolean and ordinary algebra operators. It is noted that the “gate” CON is actually a wire connection. Moreover, since each gate should be only of one type, the following constrain should hold for the gate selection binary variables $t_{sel_i}t_{sel_{l_1}} + t_{sel_{l_2}} + t_{sel_{l_3}} + t_{sel_{l_4}} + t_{sel_{l_5}} + t_{sel_{l_6}} + t_{sel_{l_7}} = 1$.

- Every gate may accept inputs from a set of possible entities $T_i$, which could either be (i) outputs of gates of the previous level, (ii) the function’s variables, or (iii) a 1 or 0 constant. Hence, for each input $inp_i$, an expression of the following form can be written:

  $$inp_i = inp_{sel_{l_1}} \times T_1 + inp_{sel_{l_2}} \times T_2 + inp_{sel_{l_3}} \times T_3 + ...$$

  where $inp_{sel_{l_j}}$ is a Boolean selection coefficient for each entity, respectively. It means that entity $T_j$ will be selected to input $inp_i$ if $inp_{sel_{l_j}} = 1$ or not if $inp_{sel_{l_j}} = 0$.

Given that an input can be fed by a unique entity, the following constrain applies for every input:

  $$inp_{sel_{l_1}} + inp_{sel_{l_2}} + inp_{sel_{l_3}} + ... = 1.$$
- Each gate’s output can be written as \( \text{out}_i = E_i \times f_i \), where \( f_i \) is the corresponding function, given its gate type, which takes as arguments its data inputs. The binary coefficient \( E_i \) indicates whether the corresponding gate will be active or not. Every inactive module can be eliminated.

Given the aim of minimizing the number of used gates, the expression \( E_1 + E_2 + E_3 + \ldots \) needs to be minimized. It is noted that the “gates” corresponding to the operation \( \text{CON} \) are not taken into account in the previous sum, since it corresponds to a simple wire connection.

- Finally, the output(s) of the root gate(s) should produce the given function(s) \( f_{\text{output}} \). Hence, the following expression(s) are defined:
  - \( O_n = 1 \), in case function \( f \) covers the examined minterm or
  - \( O_n = 0 \)

All the above described relations, for each minterm, constitute the integer non-linear problem to be solved. In the following section, an example is presented in order to clarify the proposed non-linear integer programming methodology.

4 | AN ILLUSTRATIVE EXAMPLE

Given the three-variable function \( f = \sum (0,1,3,5,6) = 01101011 = 6b \), which has minterms with representative numbers 0, 1, 3, 5, 6. Moreover, the representative bits of these minterms are as follows:

- \( b_{0,1} = 0 \) for minterm \( 0_{10} = (000)_2 \) and variable 1
- \( b_{0,2} = 0 \) for minterm \( 0_{10} = (000)_2 \) and variable 2
- \( b_{0,3} = 0 \) for minterm \( 0_{10} = (000)_2 \) and variable 3
- \( b_{1,1} = 0 \) for minterm \( 1_{10} = (001)_2 \) and variable 1
- \( \ldots \)
- \( b_{7,3} = 1 \) for minterm \( 7_{10} = (111)_2 \) and variable 3

For this example, we examine gates that can take as input:

- the previous level’s output (with the exception of the first level)
- either constant 1 or 0
- the variable of the function

These gates can be placed on a grid formation of \( 3 \times 2 \) to implement the given function. Considering the gate \( (i,j) \), we can construct the following equations for each of its two inputs \( \text{inp}1_{i,j,g} \) and \( \text{inp}2_{i,j,g} \), its output \( \text{out}_{i,j,g} \), and for each minterm \( g \) of the function to be implemented:

\[
\text{inp}1_{i,j,g} = \text{inpsel}_{i,j,1} \times \text{out}_{i-1,1,g} + \text{inpsel}_{i,j,2} \times \text{out}_{i-1,2,g} + \text{inpsel}_{i,j,3} \times b_{g,1} + \text{inpsel}_{i,j,4} \times b_{g,2} + \text{inpsel}_{i,j,5} \times b_{g,3} + \text{inpsel}_{i,j,6} \times 1 + \text{inpsel}_{i,j,7} \times 0
\]

The above expression signifies that gate \( (i,j) \) takes as first input either:

- one output of the previous level’s two gates or
- one of the three variables, for which the minterm \( g \) will have the values \( b_{g,1}, b_{g,2}, b_{g,3} \), respectively, or
- one of the constants 0 or 1

Clearly, for all the \( b’s \) that are equal to zero in the given function, the corresponding term in the above expression can be eliminated. The choice between all entities that will feed the input is made by the selection parameters \( \text{inpsel}_{i,j,k} \); hence, only one of these can be 1 while all the rest have to be 0. Mathematically, this can be expressed by the following relation:

\[
\text{inpsel}_{i,j,1} + \text{inpsel}_{i,j,2} + \text{inpsel}_{i,j,3} + \text{inpsel}_{i,j,4} + \text{inpsel}_{i,j,5} + \text{inpsel}_{i,j,6} + \text{inpsel}_{i,j,7} = 1
\]
It is noted that all the above variables represent binary ones. We can write corresponding equations for the input \( inp_{2,i,j} \):

\[
\begin{align*}
inp_{2,i,j} &= \text{inpsel}_{2,i,1} \times \text{out}_{i-1,1,1} + \text{inpsel}_{2,i,2} \times \text{out}_{i-1,1,2} + \text{inpsel}_{2,i,3} \times b_{g,1} + \text{inpsel}_{2,i,4} \times b_{g,2} + \text{inpsel}_{2,i,5} \\
& \quad \times b_{g,3} + \text{inpsel}_{2,i,6} \times 1 + \text{inpsel}_{2,i,7} \times 0 \\
\text{inpsel}_{2,i,1} + \text{inpsel}_{2,i,2} \text{inpsel}_{2,i,3} + \text{inpsel}_{2,i,4} + \text{inpsel}_{2,i,5} + \text{inpsel}_{2,i,6} + \text{inpsel}_{2,i,7} &= 1
\end{align*}
\]

Obviously, at the first level in the expressions for the inputs, the first two terms should be eliminated, since there is no previous level to provide outputs.

As far as the output \( \text{out}_{i,j,g} \), we must ensure that the gate \((i,j)\):

- will be one of a given set of gate types (described in the previous section)
- can be (potentially) eliminated
- will produce at the top level of the examined architecture the desired function

Hence, we can write the following equation for each minterm \( g \):

\[
\begin{align*}
\text{out}_{i,j,g} = E_{ij} \times (\text{inpsel}_{i,j,1} \times (\text{inpsel}_{1,j,g} \times \text{inpsel}_{2,j,g}) + \\
+ \text{outsel}_{i,j,2} \times (\text{inpsel}_{1,j,g} + \text{inpsel}_{2,j,g} - \text{inpsel}_{1,j,g} \times \text{inpsel}_{2,j,g}) + \\
+ \text{outsel}_{i,j,3} \times (1 - \text{inpsel}_{1,j,g}) + \\
+ \text{outsel}_{i,j,4} \times (\text{inpsel}_{1,j,g} + \text{inpsel}_{2,j,g} - 2 \times \text{inpsel}_{1,j,g} \times \text{inpsel}_{2,j,g}) + \\
+ \text{outsel}_{i,j,5} \times ((1 - \text{inpsel}_{1,j,g}) \times (1 - \text{inpsel}_{2,j,g}) - 2 \times (1 - \text{inpsel}_{1,j,g}) \times (1 - \text{inpsel}_{2,j,g})) + \\
+ \text{outsel}_{i,j,6} \times ((1 - \text{inpsel}_{1,j,g}) \times (1 - \text{inpsel}_{2,j,g}) + \\
+ \text{outsel}_{i,j,7} \times (\text{inpsel}_{1,j,g}))
\end{align*}
\]

The above equation describes that a gate can only be one of the possible functions \( f_k \), described in the previous section, determined by the selection variables \( \text{outsel} \). It is noted that we can use a desired subset of the above gates, omitting the appropriate lines in the above equation. Moreover, variable \( E_{ij} \) defines if the gate \((i,j)\) will be active \((E_{ij} = 1)\) or inactive \((E_{ij} = 0)\) in which case the gate can be eliminated. Hence, two more equations should be added for each minterm \( g \), in order to ensure the above requirements:

\[
\begin{align*}
\text{outsel}_{i,j,1} + \text{outsel}_{i,j,2} + \text{outsel}_{i,j,3} + \text{outsel}_{i,j,4} + \text{outsel}_{i,j,5} + \text{outsel}_{i,j,6} + \text{outsel}_{i,j,7} &= 1
\end{align*}
\]

\[
\begin{align*}
\text{Obj} = E_{1,1} \times (1 - \text{outsel}_{1,1,1}) + E_{1,2} \times (1 - \text{outsel}_{1,2,1}) + E_{2,1} \times (1 - \text{outsel}_{2,1,1}) + E_{2,2} \times (1 - \text{outsel}_{2,2,1}) + E_{3,1} \times (1 - \text{outsel}_{3,1,1}) + E_{3,2} \times (1 - \text{outsel}_{3,2,1})
\end{align*}
\]

\( \text{Obj} \) in the above equation is the objective function to be minimized, so that the produced circuit will have the minimum number of gates. \( \text{Obj} \) is an integer variable and is equal to the sum of all \( E_{ij} \)s, which are binary variables indicating that a gate is active. It is multiplied by \((1 - \text{outsel}_{i,j,7})\) for each gate \((i,j)\). This is because in case the gate \((i,j)\) is a simple wire connection, it is not counted in the cost of the circuit.

Finally, in order to make sure that the proposed circuit produces the same truth table, the following equations should be added for each minterm \( g \): \( \text{out}_{3,1,g} = 1 \) or \( \text{out}_{3,1,g} = 0 \), depending on whether function \( f \) covers or not the minterm \( g \).

The search for a solution based on all the above equations establishes the non-linear problem for the specific illustrative example. The final circuit that results from the solution is shown in the left output of Figure 1 and consists of four gates.

In case we had a two-output function with outputs \( 6b \) and \( 2a \), we would have to add the corresponding expressions for \( \text{out}_{3,2,g} \), that is, for the top level (output) gate 3,2. This final circuit (see right output of Figure 1) consists of five gates.
The proposed method can also tackle with the more difficult case of expressions whose truth table includes do-not-cares; that is, for specific minterms, we do not care for the binary value of the outputs. These functions are defined as incompletely specified functions and can be described by the use of two disjoint sets:

- on-set, with members all the minterms that drive the function to 1
- do-not-care set (dc-set), with members all the do-not-care minterms

A third set, the off-set, may be defined with members all remaining minterms not belonging to either of the previous two sets. If the above two sets are represented in a bit vector form, then the on-function and dc-function are produced.

According to our methodology, all previously mentioned expressions that incorporate do-not-care minterms are omitted; for example, if the minterms with representative numbers 0 and 1 are do-not-care ones, that is, $\bar{x}_1x_2x_3$ and

---

**FIGURE 1** Circuit of functions $f_1 = 6b$ and $f_2 = 2a$

**FIGURE 2** Circuit of functions $f_1 = 6b, f_2 = 2a$, and DCs

---

5 | INCOMPLETELY SPECIFIED FUNCTIONS
$x_1x_2x_3$, then all equations with $g = 0$ or $g = 1$ are skipped, for example, output$_{i,1} = 1$ of the previous illustrative example. Figure 2 represents the proposed architecture that requires three gates for the case of the two functions $6b$ and $2a$ including do-not-cares.

6 | IMPLEMENTATION

All our experiments were executed on the free access NEOS server.\textsuperscript{8–11} We mainly used the BARON solver,\textsuperscript{12} which determines the global optima by using branch-and-bound deterministic algorithms.

This solver requires fairly general assumptions, which are fulfilled in our case. Therefore, we are able to detect an exact solution, which provides the minimum number of gates.

| Example function in HEX | Results of reference$^x$ | Results of our method gates | Allowable gates in the circuit |
|------------------------|--------------------------|----------------------------|-------------------------------|
| 0ee9                   | Reference$^3$ $\rightarrow$ 11 | 8*                         | All gates                     |
| $\sum (0, 3, 5, 6, 7, 9, 10, 11)$ | example 11.2 |                             |                               |
| a7f1                   | Reference$^3$ $\rightarrow$ 5 | 5                           | All gates                     |
| 5a5a                   | Reference$^3$ $\rightarrow$ 7 | 7*                         | All gates                     |
| 936c                   | Reference$^3$ $\rightarrow$ 7 | 7*                         | All gates                     |
| ec80                   | Reference$^3$ $\rightarrow$ 7 | 7*                         | All gates                     |
| a0a0                   | Reference$^3$ $\rightarrow$ 7 | 7*                         | All gates                     |
| 6ac0                   | Reference$^3$ $\rightarrow$ 7 | 7*                         | All gates                     |
| 4c00                   | Reference$^3$ $\rightarrow$ 7 | 7*                         | All gates                     |
| 8000                   | Reference$^3$ $\rightarrow$ 7 | 7*                         | All gates                     |
| 25cb                   | Reference$^3$ $\rightarrow$ 7 | 7*                         | AND, OR, XOR, NOT, CON        |
| a7f1                   | Reference$^3$ $\rightarrow$ 7 | 6*                         | AND, OR, XOR, NOT, CON        |
| ab                     | Reference$^4$ $\rightarrow$ 5 | 5                           | NAND                          |
| $\sum (0, 1, 3, 5, 7)$ | Reference$^4$ $\rightarrow$ 13 | 12                         | NAND                          |
| 69                     | Reference$^4$ $\rightarrow$ 9 | 9                           | NAND                          |
| $\sum (0, 3, 5, 6)$    | Reference$^4$ $\rightarrow$ 9 | 9                           | NAND                          |
| 4a6a                   | Reference$^4$ $\rightarrow$ 9 | 9                           | NAND                          |
| $\sum (1, 3, 5, 6, 9, 11, 14)$ | Reference$^4$ $\rightarrow$ 9 | 9                           | NAND                          |
| 22d5                   | Reference$^4$ $\rightarrow$ 9 | 9                           | NAND                          |
| $\sum (0, 2, 4, 6, 7, 9, 13)$ | Reference$^4$ $\rightarrow$ 8 | 5                           | NAND, NOR                    |
| aaaaaaaa8              | Reference$^{13}$ $\rightarrow$ 8 | 5                           | NAND, NOR                    |
| $\sum (3, 5, 7, ..., 29, 31)$ | Reference$^{13}$ $\rightarrow$ 8 | 5                           | NAND, NOR                    |
| 96                     | Reference$^{14}$ $\rightarrow$ >12 | 12*                       | NAND                          |
| $\sum (1, 2, 4, 7)$   | Reference$^{14}$ $\rightarrow$ >6 | 6                           | NAND                          |
| e8                     | Reference$^{14}$ $\rightarrow$ >6 | 6                           | NAND                          |
| $\sum (3, 5, 6, 7)$   | Reference$^{14}$ $\rightarrow$ >6 | 6                           | NAND                          |
| baf c                  | Reference$^{14}$ $\rightarrow$ >7 | 7                           | NAND                          |
| $\sum (2, 3, 4, 5, 6, 7, 9, 11, 12, 13, 15)$ | Reference$^{14}$ $\rightarrow$ >7 | 7                           | NAND                          |
The free access service of NEOS limits the maximum computing time to 8 h, which was not enough for some of our examples to run in full. Hence, for these cases, the best solution was not surely found, but instead the one within the free access time limits of NEOS.

The nomenclature used by the NEOS server is that of the GAMS and AMPL formal languages. In order to ease the production of the required GAMS equivalent programs, a program in FORTRAN was implemented to automatically produce them. This FORTRAN program takes as input all necessary parameters as well as the function of interest.

We tested the cases of references\textsuperscript{2–4} using two-input gates against our approach. For all tests, our approach resulted in architectures of fewer or equal gates. All the above mentioned comparisons are shown in Table 1, where those examples that exhausted the maximum allowed running time of the NEOS system are indicated with a star character, for example, example functions 0ee9, 5a5a, 936c. However, even for such cases without the guarantee for optimal solutions, the results were the same or better. For those examples, where no star character is shown, the optimizer terminated within the maximum provided time and hence produced the optimal solution, for example, example functions a7f1, ab, 4a6a.

\section*{Conclusions}

In what was shown in the previous sections, the proposed approach is a non-linear one that can be applied for designing multi-function, multi-level, two-input multi-gates logic circuits. Based on the presented experimental results, this method outperforms other methods available in the literature, while guaranteeing minimality. Although the outperformance could be considered not that significant, the fact that our method guarantees minimality is important in some critical applications, for example, space reduction. In addition to the above, the rest contributions of this work are that it is general and flexible and can solve design problems with any predefined gates and gate topology, for example, cascade or triangular for a single-output function and orthogonal $m \times n$ for multi-output functions, and it is a novel approach which may be adopted easily to solve other kinds of logic design problems. Moreover, it can tackle with functions including do-not-care minterms and is flexible to allow the user to define the number of levels and the number of gates per level. Due to its generality, the method can be extended to use more complicated modules, instead of simple gates, and to also support multi-input gates.

The presented experimental results of Section 6 certify that large cases ($10^2$–$10^3$ unknown variables) are manageable computational wise. The latter is a great problem for such non-linear integer programming problems, which the proposed method overcomes.

Usually, logic synthesizing tools used in the industry are constrained by a very limited number of gate types (two to four) and the proposed solutions are based on heuristic methods, for example, espresso\textsuperscript{15} and exorcism.\textsuperscript{16} On the contrary, the proposed methodology allows the selection of all possible gate types and moreover gives optimal solutions. This advantage leads to a smaller number of used gates. The main contribution of this work is the novel methodology, which is totally different to known and established design methods, and furthermore, it provides exact solutions while giving the flexibility to the user to choose among all gate types. It is based on optimization tools, which are rapidly evolving, and thus, soon the existing difficulty of solving very large real life problems will be solved. Finally, our methodology may easily be used in other new domains with additional demanding limitations like reversible or quantum circuits design.

Our future endeavor will be to use this method for multi-input gates, as well as other more complicated modules, for example, for ESCTs (exclusive or complex terms).\textsuperscript{17}

\textbf{DATA AVAILABILITY STATEMENT}

Data sharing is not applicable to this article as no datasets were generated or analyzed during the current study.

\textbf{REFERENCES}

1. Sasao T. \textit{Switching Theory for Logic Synthesis}: Springer Science & Business Media; 2012.
2. Anjomshoa M, Mahani A, Beig ME. Evolutionary design and optimization of digital circuits using imperialist competitive algorithm. \textit{Int J Comput Appl.} 2011;32(1):14-19.
3. Karakatic S, Podgorelec V, Hericko M. Optimization of combinational logic circuits with genetic programming. \textit{Elektronika ir Elektrotechnika.} 2013;19(7):86-89.
4. Rajaei A, Houshmand M, Rouhani M. Optimization of combinational logic circuits using NAND gates and genetic programming. Soft Computing in Industrial Applications: Springer; 2011:405-414.

5. Coello CAC, Christiansen AD, Aguirre AH. Use of evolutionary techniques to automate the design of combinational circuits. Int J Smart Eng Syst Des. 2000;2:299-314.

6. Pavlatos C, Dimopoulos AC, Papakonstantinou G. Logic design using modules and nonlinear integer programming. J Circ Syst Comput. 2020;29(10):2050164.

7. Pavlatos C, Dimopoulos A, Papakonstantinou G. Multi-output, multi-level, NAND-gate design using non-linear programming. In: ICCOLD 2021: International Conference on Computer Organization and Logic Design; 2021; Zurich, Switzerland.

8. Coello CAC, Christiansen AD, Aguirre AH. Use of evolutionary techniques to automate the design of combinational circuits. Int J Smart Eng Syst Des. 2000;2:299-314.

9. Pavlatos C, Dimopoulos AC, Papakonstantinou G. Logic design using modules and nonlinear integer programming. J Circ Syst Comput. 2020;29(10):2050164.

10. Gropp W, Moruré JJ. Optimization environments and the NEOS server. Technical report, IL (United States), Argonne National Lab.; 1997.

11. Zhou K, Kilinci MR, Chen X, Sahinidis NV. An efficient strategy for the activation of MIP relaxations in a multicore global MINLP solver. J Global Optim. 2018;70(3):497-516.

12. Baranov S, Karatkevich A. On transformation of a logical circuit to a circuit with NAND and nor gates only. Int J Electron Telecommun. 2018;64(3):373-378.

13. Bhattacharyya M, Bhattacharyya M. A novel design approach of boolean functions with 2-input universal NAND gates using $\mu$-graph method. J Multiple-Valued Logic Soft Comput. 2008;14:177-190.

14. Brayton RK, Hachtel GD, McMullen C, Sangiovanni-Vincentelli A. Logic Minimization Algorithms for VLSI Synthesis, Vol. 2: Springer Science & Business Media; 1984.

15. Song N, Perkowski MA. EXORCISM-MV-2: minimization of exclusive sum of products expressions for multiple-valued input incompletely specified functions. In: Proceedings of the Twenty-Third International Symposium on Multiple-Valued Logic IEEE; 1993: 132-137.

16. Voudouris D, Sampson M, Papakonstantinou G. Exact ESCT minimization for functions of up to six input variables. Integration. 2008; 41(1):87-105.

How to cite this article: Dimopoulos AC, Pavlatos C, Papakonstantinou G. Multi-output, multi-level, multi-gate design using non-linear programming. Int J Circ Theor Appl. 2022;50(8):2960-2968. doi:10.1002/cta.3300