Optimal and Heuristic Algorithms to Synthesize Lattices of Four-Terminal Switches[☆]

M. Ceylan Morgül and Mustafa Altun

Department of Electronics and Communication Engineering, Istanbul Technical University, Istanbul, Turkey, 34469.

Abstract

In this work, we study implementation of Boolean functions with nano-crossbar arrays where each crosspoint behaves as a four-terminal switch controlled by a Boolean literal. These types of arrays are commonly called as *switching lattices*. We propose optimal and heuristic algorithms that minimize lattice sizes to implement a given Boolean function. The algorithms are mainly constructed on a technique that finds Boolean functions of lattices having independent inputs. This technique works recursively by using transition matrices representing columns and rows of the lattice. It performs symbolic manipulation of Boolean literals as opposed to using truth tables that allows us to successfully find Boolean functions having up to 81 variables corresponding to a 9×9-lattice. With a Boolean function of a certain sized lattice, we check if a given function can be implemented with this lattice size by defining the problem as a satisfiability problem. This process is repeated until a desired solution is found. Additionally, we fix the previously proposed algorithm that is claimed to be optimal. The fixed version guarantees optimal sizes. Finally, we perform synthesis trials on standard benchmark circuits to evaluate the proposed algorithms by considering lattice sizes and runtimes in comparison with the recently proposed three algorithms.

Keywords: Nano-crossbar Arrays, Switching Lattices, Logic Synthesis, Satisfiability.

1. Introduction

Nano-crossbar arrays have emerged as area and power efficient structures with an aim of achieving high performance computing beyond the limits of current CMOS [1, 2, 3]. Computing is achieved with crosspoints behaving as switches, either two-terminal or four-terminal. This is illustrated in Figure 1. Depending on the used technology, a two-terminal switch behaves as a diode [4, 5], a resistive/memristive switch [6], or a field effect transistor (FET) [7]. Diode and resistive switches correspond to the crosspoint structure in Figure 1 a); here, the switch is controlled by the voltage difference between the terminals. Figure 1 b) shows a FET based switch; here, the red line represents the controlling input. A four-terminal switch is given in Figure 1 c). The controlling input, not shown in the figure, has a separate physical formation from the crossbar that is thoroughly explained for different technologies in [8].

Comparing the array sizes to implement a given Boolean function, we see that the four-terminal switch based arrays overwhelm the two-terminal based ones [9]. In these comparisons resistive/memristive arrays are not taken into account. However, it is not difficult to guess that their sizes are much worse than those of diode and FET based arrays. The reason is that resistive arrays use a minterm/maxterm representation of a given Boolean function such that each minterm/maxterm is implemented by a crossbar line [6, 10, 11]. On the other hand, diode and FET based arrays do not have such restriction, so the minimal sum of product forms can be used with each product implemented by a line [7, 12, 9]. As a result, four-terminal switch based arrays offer an important size advantage. Indeed, this is not surprising since they use two dimensional paths to implement products of a given function as opposed to using one dimensional paths (crossbar lines). In this study, we further investigate four-terminal switch based arrays to synthesize Boolean functions. These types of arrays are commonly called as *switching lattices*; we use this naming throughout the paper.

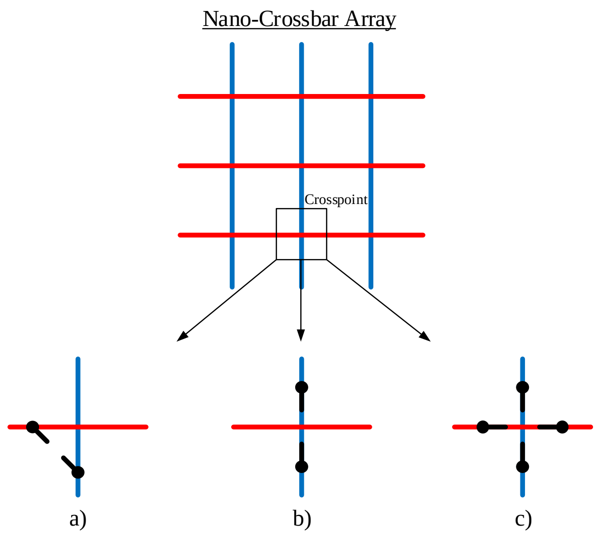


Figure 1: Switching models of a nano-crossbar array: crosspoint as a) two-terminal switch with terminals in the crossed lines, b) two-terminal switch with terminals in the same line, and c) four-terminal switch.

[☆]This work is supported by the EU-H2020-RISE project NANOxCOMP #691178 and the TUBITAK-Career project #113E760.

Email address: `{morgul, altunmus}@itu.edu.tr` (M. Ceylan Morgül and Mustafa Altun)

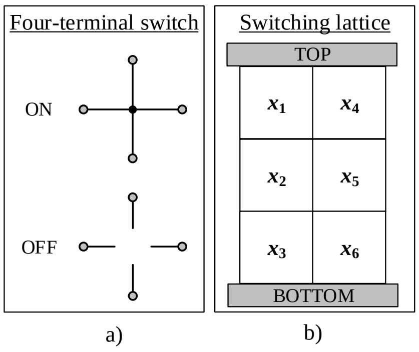


Figure 2: a) four-terminal switch, and b) switching lattice.

A four-terminal switch is shown in Figure 2 a). The four terminals of the switch are all either mutually connected (ON) or disconnected (OFF). A 3 ×2 switching lattice having 6 four-terminal switches is shown in Figure 2 b). Here, each switch is controlled by a Boolean literal. If the literal takes the value 1 (0) then the corresponding switch is ON (OFF). The Boolean function for the lattice evaluates to 1 iff there is a closed path between the top and bottom plates of the lattice. The function is obtained by taking the sum of the products of the literals along each path. These products are $x_1x_2x_3$, $x_1x_2x_5x_6$, $x_4x_5x_2x_3$, and $x_4x_5x_6$. We conclude that this lattice of four-terminal switches implements the Boolean function $x_1x_2x_3+x_1x_2x_5x_6+x_2x_3x_4x_5+x_4x_5x_6$.

The logic synthesis problem of switching lattices is first introduced in [8]. In this work, a systematic technique is proposed to implement a given Boolean function with an $m \times n$ lattice where n and m are the number of products of the function and its dual, respectively. Although it is a general and a straightforward technique, the resulting lattices may become quite large, far from optimal lattice sizes. To achieve smaller sizes, a Boolean decomposition based technique is proposed [13]. However, it is only applicable for parity functions (XOR functions). More comprehensive decomposition based studies are proposed in [14] and [15] by exploiting P-circuits and dimension-reducibility, respectively. The results are satisfactory with affordable runtimes, but still no guarantee of being close to optimal results. Furthermore, dimension-reducibility can not be applicable to all Boolean functions; there are restrictions. Another decomposition based technique is proposed for a special class of "regular" Boolean functions, called autosymmetric functions [16]. In this work, the idea of connecting separate lattices, not necessarily using a single lattice, is also examined. Although, using separate lattices can significantly reduce the total lattice area, it certainly kills the main motivation of using nano-crossbar arrays that is "overcoming interconnection problems between separate blocks/gates/transistors of conventional circuits".

There are also studies aiming at optimal results. A simple, truth table based brute-force algorithm is presented in [13]. However, as expected it suffers from high runtimes that quickly grow beyond practical limits with an increase in lattice size. Another optimal algorithm is proposed in [17]. It is an anytime algorithm that reduces the problem into a satisfiability problem with using dichotomic search. Although its runtimes are much better than those of the above mentioned one, speed of the algorithm is still an issue especially for relatively large benchmarks. Additionally, the algorithm is claimed to be optimal, but it is not for some cases. We fix this algorithm to guarantee optimal sizes.

Considering the mentioned shortcomings in the literature, we develop optimal and heuristic algorithms, based on a new technique that finds Boolean functions of lattices having independent inputs. For example, a Boolean function of a 3×3 lattice has 9 variables each of which is assigned to each of the 9 switches. This technique works recursively by using transition matrices representing columns and rows of the lattice. It performs symbolic manipulation of Boolean literals as opposed to using truth tables that allows us to successfully find Boolean functions having up to 81 variables corresponding to a 9×9 lattice. After having a Boolean function of a certain sized lattice, we check if a given target function can be implemented with this size by defining the problem as a Boolean satisfiability (SAT) problem, and using a SAT solver. This process is repeated until a desired solution is found.

Outline of the paper is as follows. In Section 2, we introduce preliminaries for switching lattices and their logic synthesis fundamentals. In Section 3, we present our optimal and heuristic algorithms. In Section 4, we first show how to fix the previously proposed optimal algorithm in [17], and then we give experimental results to evaluate the proposed algorithms by considering lattice sizes and runtimes in comparison with the recently proposed three algorithms. Section 5 concludes this study with insights and future directions.

2. Preliminaries

We first explain key concepts used in this study, and then define the logic synthesis problem with examples.

2.1. Definitions

Definition 1. *Consider k independent* **Boolean variables***, x_1, x_2, ..., x_k.* **Boolean literals** *are Boolean variables and their complements, i.e., $x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_k, \bar{x}_k$.*

Definition 2. *A* **product (P)** *is an AND of literals, e.g., $P = x_1\bar{x}_3x_4$. A* **sum-of-products (SOP)** *expression is an OR of products.*

Definition 3. *A* **sum (S)** *is an OR of literals, e.g., $S = x_1 + \bar{x}_3 + x_4$. A* **product-of-sums (POS)** *expression is an AND of sums.*

Definition 4. *A* **prime implicant (PI)** *of a Boolean function f is a product that implies f such that removing any literal from the product results in a new product that does not imply f.*

Definition 5. *An* **irredundant sum-of-products (ISOP)** *expression is an SOP expression, where each product is a PI and no PI can be deleted without changing the Boolean function f represented by the expression.*

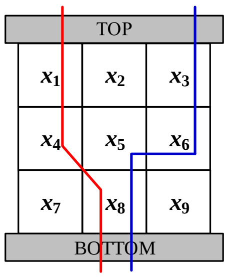


Figure 3: A lattice with eight-connected paths.

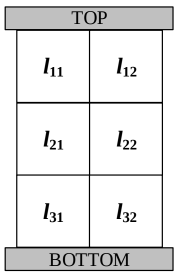


Figure 4: A 3×2 lattice.

Definition 6. *Given a Boolean function f in SOP form, let the* **degree** *of f, denoted by $degree_f$, be the maximum number of literals in a product of f.*

For example, if $f = x_1x_2x_3 + \bar{x}_1x_4$ then $degree_f = 3$.

Definition 7. *f and g are* **dual Boolean functions** *iff*

$$f(x_1, x_2, \dots, x_k) = \bar{g}(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_k).$$

Given an expression for a Boolean function in terms of AND, OR, NOT, 0, and 1, its dual can also be obtained by interchanging the AND and OR operations as well as interchanging the constants 0 and 1.

For example, if $f(x_1, x_2, x_3) = x_1x_2+\bar{x}_1x_3$ then $f^D(x_1, x_2, x_3) = (x_1 + x_2)(\bar{x}_1 + x_3)$. A trivial example is that for $f = 1$, the dual f^D is 0.

Definition 8. *An* **eight-connected path** *in a lattice, consists of both directly and diagonally adjacent sites.*

An example is shown in Figure 3. Here the paths $x_1x_4x_8$ and $x_3x_6x_5x_8$ shown by red and blue lines are both eight-connected paths; however only the blue one is four-connected. For simplicity, we generally use "path" to refer a four-connected path.

Definition 9. *Consider an $R \times C$ lattice. A* ***lattice input*** *l_{ij} is assigned to a switch/site in the ith row and the jth column of the lattice where $1 \le i \le R$ and $1 \le j \le C$. A lattice input can be a Boolean literal, logic 0, or logic 1. The* ***lattice function*** *$f_{R\times C}(l_{11}, l_{12}, .., l_{RC})$ is defined as OR of all four-connected top-to-bottom paths.*

As an example, consider a lattice in Figure 4. Here, $f_{3\times2} = l_{11}l_{21}l_{31} + l_{11}l_{21}l_{22}l_{32} + l_{12}l_{22}l_{21}l_{31} + l_{12}l_{22}l_{32}$.

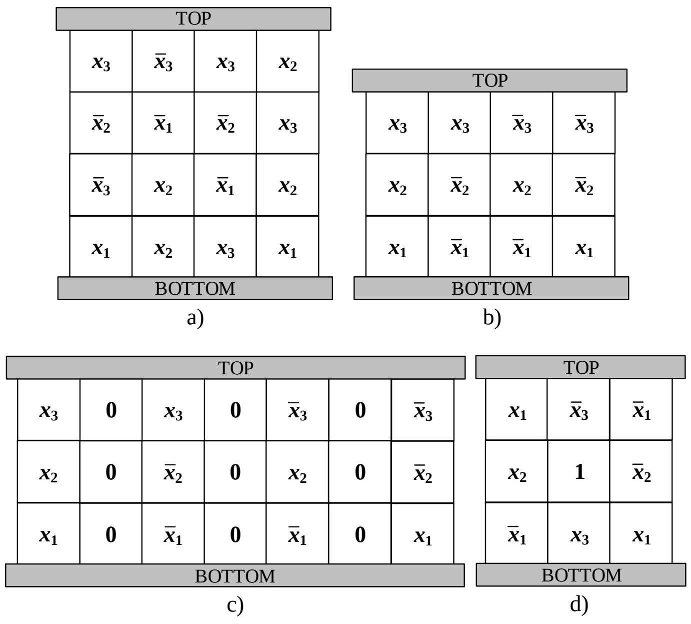

TOP
x_3 $\bar{x}_3$ x_3 x_2
$\bar{x}_2$ $\bar{x}_1$ $\bar{x}_2$ x_3
$\bar{x}_3$ x_2 $\bar{x}_1$ x_2
x_1 x_2 x_3 x_1
BOTTOM
a)
TOP
x_3 x_3 $\bar{x}_3$ $\bar{x}_3$
x_2 $\bar{x}_2$ x_2 $\bar{x}_2$
x_1 $\bar{x}_1$ $\bar{x}_1$ x_1
BOTTOM
b)
TOP
x_3 0 x_3 0 $\bar{x}_3$ 0 $\bar{x}_3$
x_2 0 $\bar{x}_2$ 0 x_2 0 $\bar{x}_2$
x_1 0 $\bar{x}_1$ 0 $\bar{x}_1$ 0 x_1
BOTTOM
c)
TOP
x_1 $\bar{x}_3$ $\bar{x}_1$
x_2 1 $\bar{x}_2$
$\bar{x}_1$ x_3 x_1
BOTTOM
d)

Figure 5: Implementation of $f_T = XOR_3$ with a) 4×4 lattice (general method in [8]), b) 4×3 lattice (method for parity functions in [8]), and c) 3×7 lattice (separation of products with 0's), and d) 3×3 lattice (optimal solution).

2.2. Synthesis Problem

Given a target Boolean function f_T, we aim to find a minimum size lattice with assigned literals, logic 0's, and logic 1's to its lattice inputs such that $f_{R\times C} = f_T$ (OR of all top-to-bottom paths equals f_T).

Suppose that $f_T = XOR_3 = x_1x_2x_3 + x_1\overline{x_2}\ \overline{x_3} + \overline{x_1}x_2\overline{x_3} + \overline{x_1}\ \overline{x_2}x_3$. Figure 5 shows different solutions to implement f_T. The first lattice in Figure 5 a) corresponds to a general method proposed in [8]. Here, R and C are found as the number of products in f_T^D and f_T, respectively, so $R = 4$, $C = 4$, and the lattice size is 16. The second one in Figure 5 b) corresponds to a specific method, only applicable for parity functions (XOR functions), again in [8]. Here, R and C are the number of variables and products in f_T, respectively, so $R = 3$, $C = 4$, and the lattice size is 12. The third one in Figure 5 c) corresponds to a general method based on separating products with 0's. Here, R is the degree of f_T and C is two times the number of products in f_T minus one, so $R = 3$, $C = 7$, and the lattice size is 21. Finally, Figure 5 d) shows the optimal solution with a lattice size of 9 that is found by applying the proposed optimal algorithm in this study.

3. Proposed Algorithms

We propose optimal and heuristic algorithms that minimize lattice sizes to implement a target Boolean function f_T. The general structure of the algorithms is presented in a flow chart in Figure 6. It has four steps; while Step 3 and Step 4 are the same for both of the algorithms, Step 1 and Step 2 have some differences. In Step 1 to determine the upper bound, we use three different formulas for the heuristic algorithm. On the other hand, we achieve a more strict upper bound for the optimal algorithm by first running the heuristic algorithm. In Step 2, the optimal

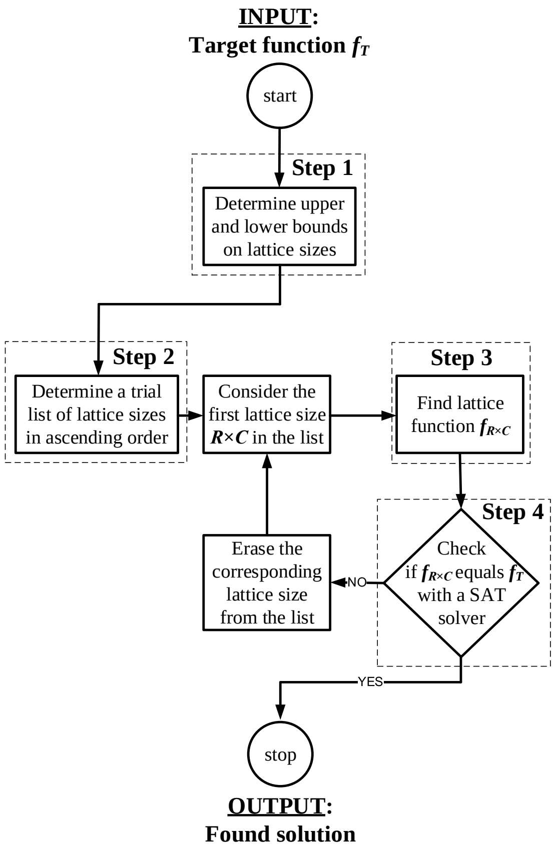

INPUT:
Target function f_T
start
Step 1
Determine upper and lower bounds on lattice sizes
Step 2
Determine a trial list of lattice sizes in ascending order
Consider the first lattice size $R\times C$ in the list
Step 3
Find lattice function $f_{R\times C}$
Step 4
Check if $f_{R\times C}$ equals f_T with a SAT solver
NO
Erase the corresponding lattice size from the list
YES
stop
OUTPUT:
Found solution

Figure 6: Flow chart of the proposed algorithms.

algorithm needs to include all probable lattice sizes into the trial list. However, only a few (or a limited number of) lattice sizes are considered for the heuristic algorithm.

In the following subsections, we elaborate on the steps followed by evaluation of the proposed algorithms.

3.1. Step 1: Upper and Lower Bounds

We directly use the lower bound (*LB*) values found in [8]. For the upper bound (*UB*), we use different approaches for the heuristic and the optimal algorithms. For the heuristic one, we consider three general implementation techniques. The first one from [8] gives lattice sizes as the number of products in f_T^D, denoted by $N_{f_T^D}$, times the number of products in f_T, denoted by N_{f_T}. As a result:

$$Lattice\_Size_1 = N_{f_T^D} \times N_{f_T}. \quad (1)$$

The second one is based on separating products of f_T with columns of 0's. Therefore,

$$Lattice\_Size_2 = degree_{f_T} \times (2 \times N_{f_T} - 1). \quad (2)$$

Finally, the third one is achieved by separating products of f_T^D by rows of 1's. Note that each product of f_T^D is implemented with a row, that corresponds to a sum for f_T, and rows of 1's makes product operations, so a product-of-sum implementation of f_T is obtained. Here,

$$Lattice\_Size_3 = (2 \times N_{f_T^D} - 1) \times degree_{f_T^D}. \quad (3)$$

We have the minimum of these three *UB* values:

$$UB = min(Lattice\_Size_1, Lattice\_Size_2, Lattice\_Size_3). \quad (4)$$

As an example, for a given f_{T1} suppose that $N_{T1} = 8$, $N_{f_{T1}^D} = 5$, $degree_{f_{T1}} = 4$, and $degree_{f_{T1}^D} = 6$. Using (4), we find that $UB = 40$. Additionally, from [8] we find that $LB = 15$.

For the optimal algorithm, we use a more strict *UB* that is achieved by running the heuristic algorithm. The found solution with a certain lattice size becomes the *UB*.

3.2. Step 2: Trial List

We have constructed the trial list according to *UB* and *LB*, sorted in ascending order. The reason of using an ascending order is that the algorithm stops when there is a solution. However, for a descending order the algorithm cannot stop when there is "no solution" for a certain sized lattice since a smaller lattice might give a solution. For example, a Boolean function can be implemented with a lattice size of 20, but cannot be implemented with a lattice size of 21. However, we can state that if both the number of rows and columns are smaller or equal to the previously used sizes, then "no solution" in the previous trial is directly applicable for the new one.

For the optimal algorithm, we consider all possible sizes between *UB* and *LB*. One thing to mention is that in forming the list we consider *LB* values not just for the lattice size, but also for the number of lattice columns and rows as given in [8]. Thus, we eliminate many trivial cases.

For the heuristic algorithm, we consider three *UB* values of $degree_{f_T}$, $N_{f_T^D}$, and $2N_{f_T^D}-1$ as well as two averages $\lfloor\frac{degree_{f_T}+N_{f_T^D}}{2}\rfloor$ and $\lfloor\frac{N_{f_T^D}+2N_{f_T^D}-1}{2}\rfloor$ for the number of rows. Similarly, we consider three *UB* values of $degree_{f_T^D}$, N_{f_T}, and $2N_{f_T} - 1$ as well as two averages $\lfloor\frac{degree_{f_T^D}+N_{f_T}}{2}\rfloor$ and $\lfloor\frac{N_{f_T}+2N_{f_T}-1}{2}\rfloor$ for the number of columns. Figure 7 illustrates these levels. As a result, there are total of 25 different lattice sizes. Since 13 of them are larger than or equal to *UB* and 4 of them are almost equal to *UB*, we only consider the remaining 8. Note that a size close to *UB* is not worth to try while we already have an *UB* solution.

3.3. Step 3: Lattice Function

We aim to find lattice functions in ISOP form. For this purpose, we need to determine paths implementing products that are not covered by products of other paths. We call this type of paths *irredundant paths*. Indeed, in general number of redundant paths in a lattice is much higher than the number of irredundant ones. Therefore eliminating redundant paths is crucial for the sake of computational time.

We propose two techniques. The first one considers paths that cannot go up, so it might calculate wrong lattice functions. On the other hand, the second one deals with all types of paths and guarantees a correct lattice function. It is fundamentally constructed on the first technique. Figure 8 shows a path having an up movement, so it is neglected by the first technique,

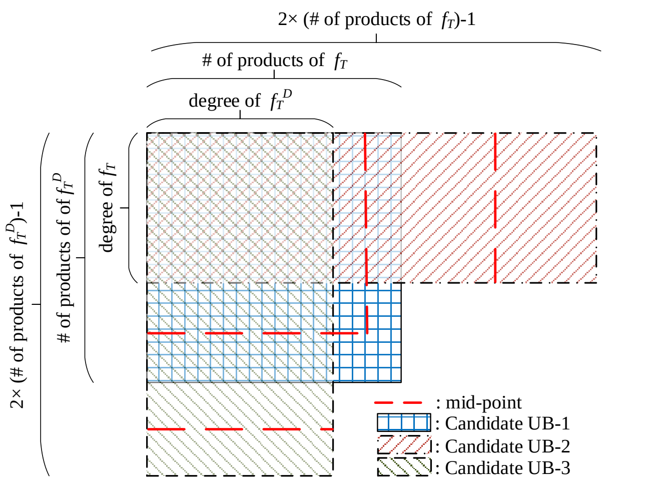
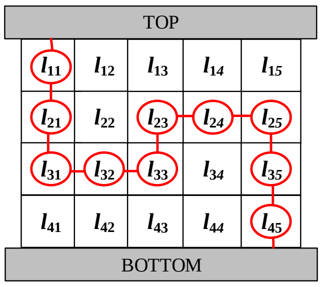


Figure 7: Illustration of the trial list of lattice sizes for the heuristic algorithm with UB candidates.



TOP
l_{11} l_{12} l_{13} l_{14} l_{15}
l_{21} l_{22} l_{23} l_{24} l_{25}
l_{31} l_{32} l_{33} l_{34} l_{35}
l_{41} l_{42} l_{43} l_{44} l_{45}
BOTTOM

Figure 8: Path with an up movement in a 4×5 lattice.

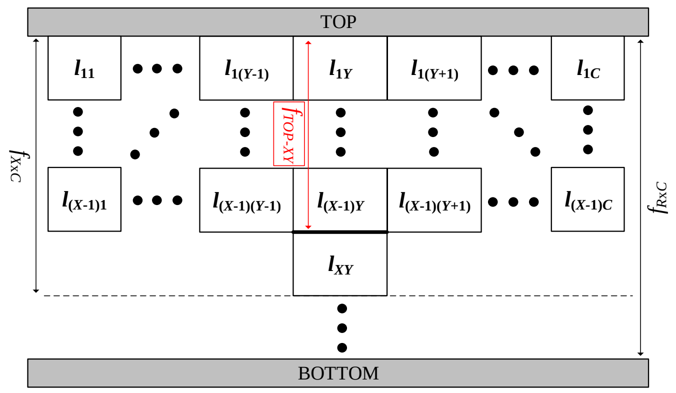


Figure 9: Illustration of f_{TOP-XY}, $f_{X \times C}$, and $f_{R \times C}$.

but accounted by the second one. Although the first one is not fully correct – we call it semi-correct, one can efficiently use it since paths having up movements are not likely being used to implement products of target functions. In terms of the computational load, the first one is much better, especially for large lattices.

For both of our algorithms, we use the second correct one.

3.3.1. *Semi-correct lattice function*

We obtain the lattice function by considering paths having left, right, and/or down movements. We only use Boolean operators (not arithmetic) for all of the following expressions.

Consider an $R \times C$ lattice. We define a dummy Boolean function f_{TOP-XY} as a sum of products of the paths between the "TOP plate" and the "upper part of the site l_{XY}".This is illustrated in Figure 9 Therefore,

$$f_{X \times C} = \sum_{Y=1}^{C} l_{XY} f_{TOP-XY}. \tag{5}$$

Recursively, we can obtain

$$f_{(X+1) \times C} = \sum_{Y=1}^{C} l_{(X+1)Y} f_{TOP-(X+1)Y} \tag{6}$$

where $f_{TOP-(X+1)Y}$ can be expresses in terms of l_{XY} and f_{TOP-XY} such that

$$\begin{aligned}
f_{TOP-(X+1)1} =& l_{X1} f_{TOP-X1} + l_{X1} l_{X2} f_{TOP-X2} + \ldots \\
& + l_{X1} l_{X2} \ldots l_{XC} f_{TOP-XC} \\
f_{TOP-(X+1)2} =& l_{X1} l_{X2} f_{TOP-X1} + l_{X2} f_{TOP-X2} + \ldots \\
& + l_{X2} l_{X3} \ldots l_{XC} f_{TOP-XC} \\
\ldots\ldots\ldots\ldots\ldots\ldots \\
f_{TOP-(X+1)C} =& l_{X1} \ldots l_{XC} f_{TOP-X1} + l_{X2} \ldots l_{XC} f_{TOP-X2} + \\
& \ldots + l_{XC} f_{TOP-XC}.
\end{aligned}$$

As a result,

$$f_{TOP-(X+1)Y} = \sum_{i=1}^{C} f_{TOP-Xi} \prod_{j=min(Y,i)}^{max(Y,i)} l_{Xj} \tag{7}$$

where $max(Y, i)$ and $min(Y, i)$ are the largest and the smallest values among Y and i, respectively.

$$\begin{bmatrix} f_{TOP-(X+1)1} \\ f_{TOP-(X+1)2} \\ \vdots \\ f_{TOP-(X+1)C} \end{bmatrix} = \begin{bmatrix} \prod_{j=1}^{1} l_{Xj} & \prod_{j=1}^{2} l_{Xj} & \cdots & \prod_{j=1}^{C} l_{Xj} \\ \prod_{j=1}^{2} l_{Xj} & \prod_{j=2}^{2} l_{Xj} & \cdots & \prod_{j=2}^{C} l_{Xj} \\ \vdots & \vdots & \ddots & \vdots \\ \prod_{j=1}^{C} l_{Xj} & \prod_{j=2}^{C} l_{Xj} & \cdots & \prod_{j=C}^{C} l_{Xj} \end{bmatrix} \cdot \begin{bmatrix} f_{TOP-X1} \\ f_{TOP-X2} \\ \vdots \\ f_{TOP-XC} \end{bmatrix}$$

Figure 10: Matrix representation of (7). [1]

Matrix representation of (7) is shown in Figure 10[1]. In this representation, if we name the column matrices having C number of $f_{TOP-(X+1)Y}$ and f_{TOP-XY} functions as F_{X+1} and F_X, respectively, and the transition matrix as $T_{(X,X+1)}$ which relates $F_{(X+1)}$ with F_X then

$$\begin{aligned}
F_{X+1} &= T_{(X,X+1)} \cdot F_X, \\
F_{X+1}^T &= F_X^T \cdot T_{(X,X+1)}, \text{and extensively} \\
F_{X+1}^T &= F_{X-1}^T \cdot T_{(X-1,X)} \cdot T_{(X,X+1)}
\end{aligned} \tag{8}$$

where $T_{(X,X+1)} = T_{(X,X+1)}^T$.

It means that we can recursively calculate F_{X+1}^T using F_1^T and related transition matrices. Since $F_1^T = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}_{1 \times C}$

[1]Matrix multiplication is denoted with "·".

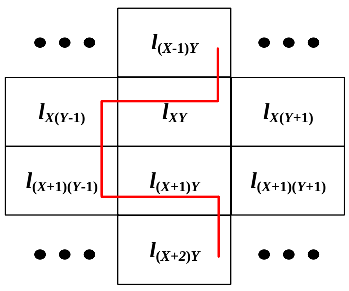


Figure 11: An example of a redundant path that needs to be eliminated.

and $F_2^T = \begin{bmatrix} l_{11} & l_{12} & \cdots & l_{1C} \end{bmatrix}$ which represents the lattice inputs for the first row, called as L_1, $F_2^T = L_1$. Similarly, we define a row matrix L_R having the lattice inputs for the last row. To conclude,

$$\begin{aligned}
f_{R\times C} &= \sum_{k=1}^{C} l_{Rk} f_{TOP-Rk}, \\
f_{R\times C} &= F_R^T \cdot L_R^T, \\
f_{R\times C} &= F_{R-1}^T \cdot T_{(R-1,R)} \cdot L_R^T, \\
f_{R\times C} &= F_{R-2}^T \cdot T_{(R-2,R-1)} \cdot T_{(R-1,R)} \cdot L_R^T, \\
&\;.... \\
f_{R\times C} &= F_1^T \cdot T_{(1,2)} \cdot T_{(2,3)} \cdots T_{(R-1,R)} \cdot L_R^T, \text{and finally} \\
f_{R\times C} &= L_1 \cdot T_{(2,R)} \cdot L_R^T.
\end{aligned} \tag{9}$$

In this calculation, it is not guaranteed that the final form of $f_{R\times C}$ is in ISOP form, so a further simplification might be needed. An example of a redundant path is given in Figure 11. This type of paths occur iff there are opposite movements in adjacent rows. To completely eliminate them, we add extra products consisting of negated inputs into the elements of transition matrices. Thus, redundant paths have both an input and its negated form, so they evaluate to 0. Excluding the elements in the matrix diagonal, we add $\overline{l_{(X-1)(i-1)}}$ for the elements in the lower triangle part, and $\overline{l_{(X-1)(i+1)}}$ for the upper triangle part; i represents the row number. This is illustrated in Figure 12. Final version of a transition matrix is given in (10):

$$T_{(X,X+1)}(k,l) = \begin{cases} \overline{l_{(X-1)(k+1)}} \prod_{j=k}^{l} l_{Xj} & l > k \\ l_{Xk} & l = k \\ \overline{l_{(X-1)(k-1)}} \prod_{j=l}^{k} l_{Xj} & l < k \end{cases} \tag{10}$$

where k and l represent row and column numbers of the matrix, respectively.

We present a few examples to elucidate our technique of obtaining lattice functions in ISOP forms.

Example 1. *Calculate $f_{3\times 3}$; $R = 3$ and $C = 3$.*

$$\begin{aligned}
L_1 &= \begin{bmatrix} l_{11} & l_{12} & l_{13} \end{bmatrix} \\
T_{(2,3)} &= \begin{bmatrix} l_{21} & l_{21}l_{22} & l_{21}l_{22}l_{23} \\ l_{21}l_{22} & l_{22} & l_{22}l_{23} \\ l_{21}l_{22}l_{22} & l_{23}l_{23} & l_{23} \end{bmatrix} \\
L_3 &= \begin{bmatrix} l_{31} & l_{32} & l_{33} \end{bmatrix} \\
f_{3\times 3} &= L_1 \cdot T_{(2,3)} \cdot L_3^T \\
f_{3\times 3} &= l_{11}l_{21}l_{31} + l_{12}l_{22}l_{32} + l_{13}l_{23}l_{33} \\
&\quad + l_{11}l_{21}l_{22}l_{32} + l_{12}l_{22}l_{23}l_{33} + l_{13}l_{23}l_{22}l_{32} \\
&\quad + l_{12}l_{22}l_{21}l_{31} + l_{11}l_{21}l_{22}l_{23}l_{33} + l_{13}l_{23}l_{22}l_{21}l_{31}
\end{aligned}$$

Example 2. *Calculate $f_{4\times 2}$; $R = 4$ and $C = 2$.*

$$\begin{aligned}
L_1 &= \begin{bmatrix} l_{11} & l_{12} \end{bmatrix} \\
T_{(2,3)} &= \begin{bmatrix} l_{21} & l_{21}l_{22} \\ l_{21}l_{22} & l_{22} \end{bmatrix} \\
T_{(3,4)} &= \begin{bmatrix} l_{31} & \overline{l_{22}}l_{31}l_{32} \\ \overline{l_{21}}l_{31}l_{32} & l_{32} \end{bmatrix} \\
L_4 &= \begin{bmatrix} l_{41} & l_{42} \end{bmatrix} \\
f_{4\times 2} &= L_1 \cdot T_{(2,3)} \cdot T_{(3,4)} \cdot L_4^T \\
f_{4\times 2} &= L_1 \cdot T_{(2,4)} \cdot L_4^T \\
f_{4\times 2} &= l_{11}l_{21}l_{31}l_{41} + l_{11}l_{21}l_{31}l_{32}l_{42} + l_{11}l_{21}l_{22}l_{32}l_{42} + \\
&\quad l_{12}l_{22}l_{32}l_{42} + l_{12}l_{22}l_{32}l_{31}l_{41} + l_{12}l_{22}l_{21}l_{31}l_{41}
\end{aligned}$$

3.3.2. *Correct lattice function*

We obtain the lattice function by considering paths having left, right, up, and/or down movements. Therefore, all types of paths are considered including paths having up movements that are neglected in calculation of semi-correct lattice functions. For this purpose, we update transition matrix elements. Each element of a transition matrix $T_{(X,X+1)}$ in (10) represents a path between two sites in the Xth row of the lattice that makes left or right movements, but no up movements. To consider up movements, we calculate a semi-correct function between the sites by transposing the related sub-matrix – later, we call this function $f_{l_{Xk}-l_{Xl}}$. Note that left and right movements in the transposed matrix correspond to down and up movements in the original matrix. The elements of the transition matrix become,

$$T_{(X,X+1)}(k,l) = \begin{cases} \overline{l_{(X-1)(k+1)}} f_{l_{Xk}-l_{Xl}} & l \geq k+4 \\ \overline{l_{(X-1)(k+1)}} \prod_{j=k}^{l} l_{Xj} & k+4 > l > k \\ l_{Xk} & l = k \\ \overline{l_{(X-1)(k-1)}} \prod_{j=l}^{k} l_{Xj} & k-4 < l < k \\ \overline{l_{(X-1)(k-1)}} f_{l_{Xk}-l_{Xl}} & l \leq k-4 \end{cases} \tag{11}$$

where k and l represent row and column numbers of the matrix, respectively. Here, a dummy Boolean function $f_{l_{Xk}-l_{Xl}}$ is used.

Note that the only difference between (10) and (11) is on the calculation of $T_{(X,X+1)}(k,l)$ where $l \geq k+4$ and $l \leq k-4$. In (11), instead of calculating a single path between l_{Xk} and l_{Xl} ($\prod_{j=k}^{l} l_{Xj}$) without an up movement as in (10), we are calculating all probable paths having all types of movements. Calculation of $f_{l_{Xk}-l_{Xl}}$ gives these paths between l_{Xk} and l_{Xl}. This function is obtained by calculating a semi-correct lattice function in Figure 13. Note that, it mainly consists of the transpose of the related part of the original lattice (Figure 9). In (11), inequalities to represent the cases are obtained by considering the

$$T_{(X,X+1)} = \begin{bmatrix} \prod_{j=1}^{1} l_{Xj} & \overline{l_{(X-1)2}}\prod_{j=1}^{2} l_{Xj} & \cdots & \overline{l_{(X-1)2}}\prod_{j=1}^{C-1} l_{Xj} & \overline{l_{(X-1)2}}\prod_{j=1}^{C} l_{Xj} \\ \overline{l_{(X-1)1}}\prod_{j=1}^{2} l_{Xj} & \prod_{j=2}^{2} l_{Xj} & \cdots & \overline{l_{(X-1)3}}\prod_{j=2}^{C-1} l_{Xj} & \overline{l_{(X-1)3}}\prod_{j=2}^{C} l_{Xj} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \overline{l_{(X-1)(C-2)}}\prod_{j=1}^{C-1} l_{Xj} & \overline{l_{(X-1)(C-2)}}\prod_{j=2}^{C-1} l_{Xj} & \cdots & \prod_{j=C-1}^{C-1} l_{Xj} & \overline{l_{(X-1)C}}\prod_{j=C-1}^{C} l_{Xj} \\ \overline{l_{(X-1)(C-1)}}\prod_{j=1}^{C} l_{Xj} & \overline{l_{(X-1)(C-1)}}\prod_{j=2}^{C} l_{Xj} & \cdots & \overline{l_{(X-1)(C-1)}}\prod_{j=C-1}^{C} l_{Xj} & \prod_{j=C}^{C} l_{Xj} \end{bmatrix}$$

Figure 12: Transition matrix formation to eliminate redundant paths.

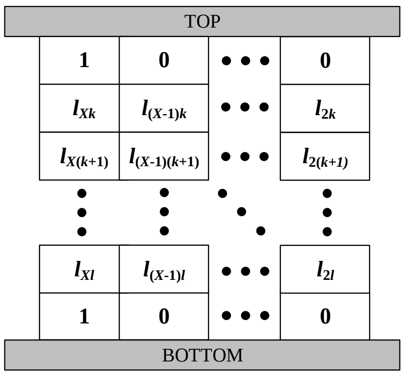


Figure 13: Lattice realizing a $f_{l_{Xk}-l_{Xl}}$ that represents the connection between l_{Xk} and l_{Xl}.

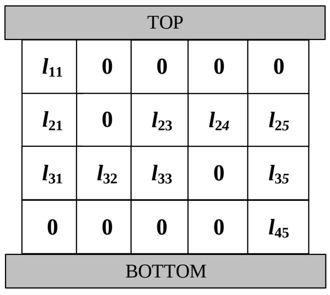

TOP
l_{11} 0 0 0 0
l_{21} 0 l_{23} l_{24} l_{25}
l_{31} l_{32} l_{33} 0 l_{35}
0 0 0 0 l_{45}
BOTTOM

Figure 14: A 4×5 lattice with a path having an up movement.

fact that the smallest lattice having up movements should have at least 4 rows and 5 columns as previously shown in Figure 8.

We present an example to elucidate our technique of obtaining correct lattice functions.

Example 3. *Calculate the correct function f of the 4×5 lattice in Figure 14 (Note that if we calculated a semi-correct function of the lattice, it would be 0).*

$$L_1 = \begin{bmatrix} l_{11} & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$T_{(2,3)} = \begin{bmatrix} l_{21} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & l_{23} & l_{23}l_{24} & l_{23}l_{24}l_{25} \\ 0 & 0 & l_{23}l_{24} & l_{24} & l_{24}l_{25} \\ 0 & 0 & l_{23}l_{24}l_{25} & l_{24}l_{25} & l_{25} \end{bmatrix}$$

$$T_{(3,4)} = \begin{bmatrix} l_{31} & \overline{l_{22}}l_{31}l_{32} & \overline{l_{22}}l_{31}l_{32}l_{33} & 0 & \overline{l_{22}}f_{l_{31}-l_{35}} \\ \overline{l_{21}}l_{31}l_{32} & l_{32} & \overline{l_{23}}l_{32}l_{33} & 0 & 0 \\ \overline{l_{22}}l_{31}l_{32}l_{33} & \overline{l_{22}}l_{32}l_{32} & l_{33} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \overline{l_{24}}f_{l_{31}-l_{35}} & 0 & 0 & 0 & l_{35} \end{bmatrix}$$

$$f_{l_{31}-l_{35}} = l_{31}l_{32}l_{33}l_{23}l_{24}l_{25}l_{35}$$

$$L_R = \begin{bmatrix} 0 & 0 & 0 & 0 & l_{45} \end{bmatrix}$$

$$f = L_1 \cdot T_{(2,3)} \cdot T_{(3,4)} \cdot L_4^T$$

$$f = L_1 \cdot T_{(2,4)} \cdot L_4^T$$

$$f = l_{11}l_{21}l_{31}l_{32}l_{33}l_{23}l_{24}l_{25}l_{35}l_{45}$$

Pseudo code of the algorithm is given below.

Algorithm Calculation of Correct or Semi-Correct Lattice Function $f_{R\times C}$

```
 1: Input: $R$: number of Rows, $C$: number of Columns, $Flag_{Correct}$: Flag indicates Correct or Semi-Correct Calculation
 2: Output: $f_{R\times C}$ function in ISOP form
 3:
 4: function CREATE_TRANS_MAT($LM, X, Flag_{Correct}$)
 5:     for $k$ in $1 : C$ do
 6:         for $l$ in $1 : C$ do
 7:             if $Flag_{Correct} == FALSE$ then
 8:                 $T_{(X,X+1)}(k,l) \leftarrow$ elements of $LM$ according to (10)
 9:             end if
10:             if $Flag_{Correct} == TRUE$ then
11:                 $T_{(X,X+1)}(k,l) \leftarrow$ elements of $LM$ according to (11) ▷ calculate $f_{l_{Xk}-l_{Xl}}$ recursively.
12:             end if
13:         end for
14:     end for
15:     return $T_{(X,X+1)}$
16: end function
17:
18: create $LM$ Matrix with elements of $l_{ij}$ with size of $R\times C$ ▷ $1 \le i \le R$, $1 \le j \le C$
19: $L_1 \leftarrow [l_{11} \cdots l_{1(C-1)} \; l_{1C}]$ (first row of $LM$)
20: $L_R \leftarrow [l_{R1} \cdots l_{R(C-1)} \; l_{RC}]$ (last row of $LM$)
21: $T2R \leftarrow$ Unit matrix
22: for X in [2,$R$-1] do
23:     $T_{(X,X+1)} \leftarrow$ CREATE_TRANS_MAT($LM, X, Flag_{Correct}$)
24:     $T2R$ = LOGIC_MATRIX_MULTIPLICATION($T2R, T_{(X,X+1)}$) ▷ Logic $OR$ and $AND$ operations used in matrix multiplications instead of conventional algebraic additions and multiplications.
25: end for
26: $temp \leftarrow$ LOGIC_MATRIX_MULTIPLICATION($L_1, T2R$)
27: $f_{R\times C} \leftarrow$ LOGIC_MATRIX_MULTIPLICATION($temp, L_R^T$)
28: Store $f_{R\times C}$
29: return $f_{R\times C}$
```

3.4. Step 4: SAT Equivalence

We use a SAT solver to check if a given target function can be implemented with a certain lattice size. First, the problem needs to be turned into a satisfiability problem using a conjunctive disjoint form (CNF) or a POS form. Since we calculate lattice functions in SOP form, and the target function is given in a PLA form that is also a SOP form, we can easily combine them into one function f_{SAT} in CNF. If f_{SAT} is satisfiable, it means that the target function f_T can be implemented with an $R{\times}C$ lattice. Core of this relation is that f_T is $TRUE$ iff $f_{R\times C}$ is $TRUE$:

$$f_T \iff f_{R\times C} \tag{12}$$

that is also used in [17]. More explicitly,

$$\begin{aligned} &a)\quad f_{R\times C} \Rightarrow f_T \quad (f_{R\times C} \bigwedge \overline{f_T}) \\ &b)\quad \overline{f_{R\times C}} \Rightarrow \overline{f_T} \quad (\overline{f_{R\times C}} \bigwedge f_T). \end{aligned} \tag{13}$$

We need both $f_{R\times C}$ and $\overline{f_{R\times C}}$ in CNF, preferably in irredundant CNF for the sake of computational time. Indeed, $f_{R\times C}$ in ISOP form is same as $\overline{f_{R\times C}}$ in irredundant CNF with negated inputs. Similarly, $\overline{f_{R\times C}}$ in ISOP form is the same as $f_{R\times C}$ in irredundant CNF with negated inputs. Therefore, along with $f_{R\times C}$ in ISOP form, we also need $\overline{f_{R\times C}}$ in ISOP form that can be computed using logic minimization tools such as Espresso [18].

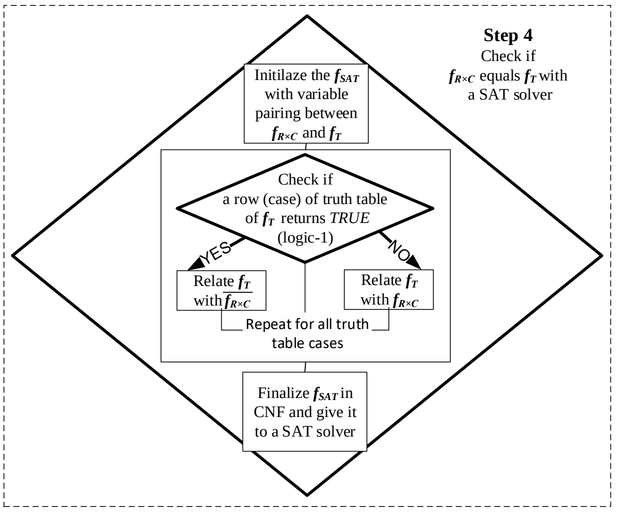
Summary of how we use SAT equivalence in Step-4 of our algorithm is given in Figure 15. As an example, assume that check whether a target function $f_T = x_1\, x_2 + x_3$ is implementable with a 2×2 lattice. Here, $f_{R\times C} = f_{2\times 2} = l_{11}l_{21} + l_{12}l_{22}$ and $\overline{f_{R\times C}} = \overline{f_{2\times 2}} = \overline{l_{11}}\,\overline{l_{12}} + \overline{l_{11}}\,\overline{l_{22}} + \overline{l_{21}}\,\overline{l_{12}} + \overline{l_{21}}\,\overline{l_{22}}$. Since f_T has three variables, there are $2^3 = 8$ truth table cases; 5 of them make $f_T = 1$ ($TRUE$) and 3 of them make $f_T = 0$ ($FALSE$). Thus, to calculate f_{SAT} in CNF form we use the relation in (13a) for the five cases and the relation in (13b) for the remaining three.

Our treatment does not fit the 3-SAT rule, since paths are directly used as SAT problem clauses. Although it is possible to turn these clauses into 3-termed clauses, this would extensively enlarge the number of clauses in the final form that causes dramatic runtime increases. In [17], they build their SAT problem with constraints considering the 3-SAT rule. However, at the end, the total number of variables and clauses are much lower for our case compared to those used in [17].

3.5. Evaluation of Algorithms

Suppose that a given function f_T and its dual f_T^D have a total of m products in their SOP form. Also suppose that f_T has n variables. Total time needed for our algorithms can be represented as (time needed to determine UB and LB in Step 1) + (number of trials of lattice sizes in Step 2)×((time needed to obtain $f_{R\times C}$ in Step 3)+(time needed for the SAT solver to check an equivalence in Step 4)).

The third and the fourth terms, corresponding to Step 3 and Step 4 of the algorithms respectively, are the same for both the



Step 4
Check if $f_{R\times C}$ equals f_T with a SAT solver
Initilaze the f_{SAT} with variable pairing between $f_{R\times C}$ and f_T
Check if a row (case) of truth table of f_T returns $TRUE$ (logic-1)
YES
NO
Relate f_T with $\overline{f_{R\times C}}$
Relate f_T with $f_{R\times C}$
Repeat for all truth table cases
Finalize f_{SAT} in CNF and give it to a SAT solver

Figure 15: Flow chart of Step-4 of the algorithm.

heuristic and the optimal algorithms. For the third term, matrix multiplications are performed to calculate $f_{R\times C}$ in ISOP form. For both the number of transition matrices and the matrix dimensions are upper bounded by m. There are $m-2$ number of transition matrices with dimensions of $m \times m$. Therefore, the time complexity becomes $O(m^m)$. If a truth table based technique was used then the total number of truth table rows would be upper bounded by $2^{(m^2)}$ and for each row to determine the output as logic 0 or 1, we would need m^2 operations. As a result the complexity would be $O(m^2 2^{(m^2)})$ (still not in ISOP form). This is much worse than what we have. Another important aspect is that the cost of obtaining $f_{R\times C}$ is mostly independent of a given function. Therefore, we can share this cost among target functions by initially creating a library of all probable $f_{R\times C}$'s.

The fourth term corresponds to a SAT solver. Here, the total time is linearly dependent on the number of truth table rows that is upper bounded by 2^n, the number of the clauses that is upper bounded by m^2, and the number of literals in a clause that is also upper bounded by m^2. As a result, the complexity is $O(m^4 2^n)$.

For the heuristic algorithm, the first term corresponding to Step 1 has a complexity of $O(1)$ since a fixed number of calculations is done. The second term also has a $O(1)$ complexity since the number of trials is upper bounded by 8. As a result, the time complexity for the whole algorithm becomes $O(m^m)$ if no library of $f_{R\times C}$'s is constructed, and $O(m^4 2^n)$ is the library is constructed.

For the optimal algorithm, the first term corresponding to Step 1 has the same complexity of the heuristic algorithm since we run the heuristic algorithm to obtain the UB. The second term corresponding to Step 2 has a $O(m^2)$ complexity since UB is upper bounded by m^2. As a result, the time complexity of the optimal algorithm is m^2 times larger than that of the heuristic algorithm.

Of course, all these analyses are based on the worst-case scenarios. Therefore, real runtimes given in the next section might be different, generally better, than what we expect.

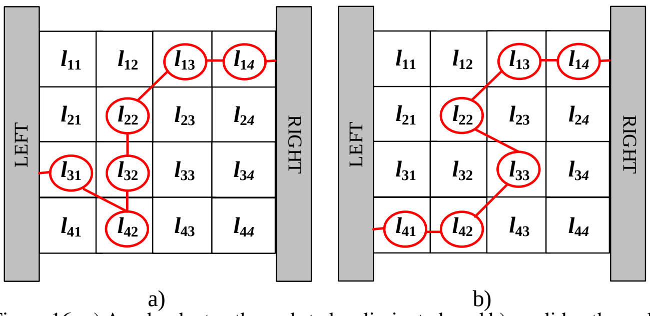

LEFT
RIGHT
l_{11} l_{12} l_{13} l_{14}
l_{21} l_{22} l_{23} l_{24}
l_{31} l_{32} l_{33} l_{34}
l_{41} l_{42} l_{43} l_{44}
a)
b)

Figure 16: a) A redundant path needs to be eliminated, and b) a valid path needs to be accounted.

4. Experimental Results

Before presenting experimental results, we first show how we fix the the optimal algorithm in [17].

4.1. Fixing the Optimal Algorithm

We fix the previously proposed algorithm in [17] that is claimed to be optimal, but it is not for some cases. The fixed version guarantees optimal sizes.

While constructing eight-connected paths between left and right plate, they put a constraint to eliminate redundant paths such as the one shown in Figure 16 a). However, this causes elimination of irredundant paths such as the one shown in Figure 16 b). The reason of this mis-elimination is their constraint definition:

- "A redundant path should have at most one element from the 2nd and the $(C-1)$th columns."

A redundant path in Figure 16 b) is a counter example for this constraint. We have corrected it as:

- "A redundant path should have a single element from the first and the last (Cth) column such that this element has a single neighbor element in the path. "

Thus, irredundant paths having more than one element in the 2nd and the $(C-1)$th columns, such as the one in Figure 16 b), are considered.

In the following part, benchmark simulation results show some cases such that the fixed algorithm gives a correct result, but the previously proposed algorithm does not.

4.2. Comparing Optimization Algorithms

We compare six different algorithms by considering runtime and lattice sizes for different benchmarks[2]. We treat each output of a benchmark circuit as a separate target function. We limit the runtime with 10800 seconds (3 hours). Among these six algorithms, three of them are previously proposed algorithms in [14], [15], and [17]; one of the them is the fixed version of the optimal algorithm in [17]; and the remaining two are the proposed optimal and heuristic algorithms. Experiments run on a 3.20GHz Intel Core i7 CPU (only single core used) with 4GB memory. Used SAT solver is *glucose*_3.0 [19].

[2]Source codes of all algorithms are available at `http://www.ecc.itu.edu.tr/images/3/33/Algorithms_for_Switching_Lattices.zip`

Results are given in Table 1 and Table 2. In Table 2, we aim to compare non-optimal algorithms with the guidance of the fixed optimal algorithm by using relatively large benchmark functions. Examining the numbers in Table 1, we see that the proposed optimal algorithm outperforms the other optimal algorithms with the best runtime for most of the cases. Also note that for three cases corresponding to the benchmarks "b12_01", "dc1_02" and "ex5_12", the optimal algorithm in [17] does not find the optimal solution, but both the fixed version of it and the proposed algorithm find the solution.

Considering the results for the non-optimal algorithms in Table 1 and Table 2, we see the superiority of the proposed heuristic algorithm offering small sizes and high speed. For 65 benchmarks out of 70, it results in optimal sizes. For example, consider "clpl_03". Algorithm "Proposed (Optimal)" finds the optimal solution in 53 seconds; "Fixed Version of [17] (Optimal)" finds in 156 seconds; and "Proposed Heuristic (non-Optimal)" finds just in 18 seconds. For couple of relatively large functions, "Proposed (Optimal)" could not find the solution inside the time limit yet other optimal algorithms do. The reason is that the proposed optimal algorithm does not fit to the 3-SAT rule but "Fixed Version of [17] (Optimal)" fits.

For relatively small number of cases, decomposition based algorithms in [14] and [15] give the best runtime values, but their solutions are generally much larger than the optimal ones. When we compare our non-optimal heuristic algorithm with them, we observe that our algorithm offers an average of 23.07% and 20.51% lattice size improvements over the alorithms in [14] and [15], respectively. The compared algorithms even yield larger sizes than the upper bound used for the proposed optimal algorithm, for the benchmark "mp2d_08". Additionally, applicability of the dimension-reducing based algorithm is quite limited.

5. Conclusion

In this study, we propose logic synthesis algorithms for switching lattices. We offer both optimal and heuristic algorithms to implement a given Boolean function with optimized lattice sizes. Our algorithms are fundamentally constructed on a technique that finds Boolean functions of lattices having independent inputs. This technique can also be used to find a Boolean function of a given lattice with assigned inputs. For our algorithms, we translate the problem of checking whether a given Boolean function can be implemented with a certain sized lattice, to the SAT problem. Our algorithms give considerably better results in terms of lattice size and runtime compared to previously proposed algorithms.

As a future work, we aim to construct multi-output lattices to implement multi-output Boolean functions. So far, the literature only considers single output lattices. Another direction is the investigation of reconfigurability in switching lattices.

References

[1] W. Lu, C. M. Lieber, Nanoelectronics from the bottom up, Nature materials 6 (11) (2007) 841–850.

[2] A. Zhang, G. Zheng, C. M. Lieber, Nanoelectronics, circuits and nanoprocessors, in: Nanowires, Springer, 2016, pp. 103–142.

[3] D. Alexandrescu, M. Altun, L. Anghel, A. Bernasconi, V. Ciriani, L. Frontini, M. Tahoori, Synthesis and performance optimization of a switching nano-crossbar computer, in: Digital System Design (DSD), 2016 Euromicro Conference on, IEEE, 2016, pp. 334–341.

[4] S. C. Goldstein, M. Budiu, Nanofabrics: Spatial computing using molecular electronics, ACM SIGARCH Computer Architecture News 29 (2) (2001) 178–191.

[5] Y. Huang, X. Duan, Y. Cui, L. J. Lauhon, K.-H. Kim, C. M. Lieber, Logic gates and computation from assembled nanowire building blocks, Science 294 (5545) (2001) 1313–1317.

[6] G. Snider, Computing with hysteretic resistor crossbars, Applied Physics A: Materials Science & Processing 80 (6) (2005) 1165–1172.

[7] G. Snider, P. Kuekes, T. Hogg, R. S. Williams, Nanoelectronic architectures, Applied Physics A 80 (6) (2005) 1183–1195.

[8] M. Altun, M. D. Riedel, Logic synthesis for switching lattices, IEEE Transactions on Computers 61 (11) (2012) 1588–1600.

[9] M. C. Morgul, M. Altun, Synthesis and optimization of switching nanoarrays, in: Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2015 IEEE 18th International Symposium on, IEEE, 2015, pp. 161–164.

[10] P. Huang, J. Kang, Y. Zhao, S. Chen, R. Han, Z. Zhou, Z. Chen, W. Ma, M. Li, L. Liu, et al., Reconfigurable nonvolatile logic operations in resistance switching crossbar array for large-scale circuits, Advanced Materials 28 (44) (2016) 9758–9764.

[11] L. Xie, H. A. Du Nguyen, M. Taouil, S. Hamdioui, K. Bertels, A mapping methodology of boolean logic circuits on memristor crossbar, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[12] T. Wang, P. Narayanan, M. Leuchtenburg, C. A. Moritz, Nasics: A nanoscale fabric for nanoscale microprocessors, in: Nanoelectronics Conference, 2008. INEC 2008. 2nd IEEE International, IEEE, 2008, pp. 989–994.

[13] M. C. Morgül, M. Altun, Anahtarlamalı nano dizinler ile lojik devre tasarımı ve boyut optimizasyonu logic circuit design with switching nano arrays and area optimization, in: ELECO, 2014.

[14] A. Bernasconi, V. Ciriani, L. Frontini, V. Liberali, G. Trucco, T. Villa, Logic synthesis for switching lattices by decomposition with p-circuits, in: Digital System Design (DSD), 2016 Euromicro Conference on, IEEE, 2016, pp. 423–430.

[15] A. Bernasconi, V. Ciriani, L. Frontini, G. Trucco, Synthesis on switching lattices of dimension-reducible boolean functions, in: Very Large Scale Integration (VLSI-SoC), 2016 IFIP/IEEE International Conference on, IEEE, 2016, pp. 1–6.

[16] A. Bernasconi, Composition of switching lattices and autosymmetric boolean function synthesis, in: Digital System Design (DSD), 2016 Euromicro Conference on, IEEE, 2017.

[17] G. Gange, H. Søndergaard, P. J. Stuckey, Synthesizing optimal switching lattices, ACM Transactions on Design Automation of Electronic Systems (TODAES) 20 (1) (2014) 6.

[18] R. K. Brayton, G. D. Hachtel, C. McMullen, A. Sangiovanni-Vincentelli, Logic minimization algorithms for VLSI synthesis, Vol. 2, Springer Science & Business Media, 1984.

[19] G. Katsirelos, A. Sabharwal, H. Samulowitz, L. Simon, et al., Resolution and parallelizability: Barriers to the efficient parallelization of sat solvers., in: AAAI, Citeseer, 2013.

Table 1: Comparison of Optimization Algorithms

	Optimal in [17] (non-Optimal)		**Proposed** (Optimal)		**Fixed** Version of [17] (Optimal)		**Proposed** Heuristic (non-Optimal)		P-Decomposition (non-Optimal) [14]		D-Reducing (non-Optimal) [15]	
Benchmark Name	Elapsed Time (s)	Size	Elapsed Time (s)	Size	Elapsed Time (s)	Size	Elapsed Time (s)	Size	Elapsed Time (s)	Size	Elapsed Time (s)	Size
alu1_00	0.003	2x3	**0.002**	**2x3**	0.008	2x3	0.004	2x3	0.081	2x3	N/A	-
alu1_01	**0.002**	**3x2**	0.005	3x2	0.004	3x2	0.004	3x2	0.031	3x2	0.04	4x2
b12_00	0.838	4x3	0.139	4x3	0.747	4x3	**0.04**	**4x3**	0.407	4x3	0.28	5x3
b12_01*	6.066	4x4	1.102	**5x3**	3.218	5x3	**0.05**	4x4	0.963	4x4	0.71	6x3
b12_02	15.395	4x4	5.440	4x4	15.949	4x4	**0.665**	**4x4**	2.118	5x8	N/A	-
b12_03	0.060	3x2	**0.022**	**3x2**	0.059	3x2	0.023	3x2	0.092	2x5	N/A	-
b12_04	0.004	2x4	**0.002**	**2x4**	0.005	2x4	0.002	2x4	0.111	2x5	N/A	-
b12_06	72.327	5x4	**	**	79.957	5x4	14.303	**5x4**	**9.035**	6x8	15.21	8x3
b12_07	2.130	3x6	6.362	3x6	2.602	3x6	**0.088**	**3x6**	1.433	5x7	N/A	-
b12_08	0.003	2x7	0.003	2x7	0.004	2x7	**0.002**	**2x7**	0.285	2x10	N/A	-
c17_00	0.064	2x3	**0.018**	**2x3**	0.064	2x3	0.022	2x3	0.048	2x4	N/A	-
c17_01	0.060	3x2	**0.019**	**3x2**	0.057	3x2	0.020	3x2	0.138	3x2	N/A	-
clpl_00	0.569	3x4	0.402	3x4	0.555	3x4	**0.042**	**3x4**	0.104	4x5	N/A	-
clpl_01	0.003	3x3	0.003	3x3	0.003	3x3	**0.002**	**3x3**	0.059	3x6	N/A	-
clpl_02	0.003	2x2	0.002	2x2	0.003	2x2	**0.002**	**2x2**	0.065	2x3	N/A	-
clpl_03	103.284	3x6	53.047	3x6	156.242	3x6	18.150	**3x6**	**3.547**	6x9	N/A	-
clpl_04	15.388	3x5	3.362	3x5	13.730	3x5	**0.158**	**3x5**	0.672	5x8	N/A	-
dc1_00	0.157	3x3	0.022	3x3	**0.148**	**3x3**	0.030	3x3	0.145	4x4	N/A	-
dc1_01	0.004	3x2	0.003	3x2	0.003	3x2	**0.002**	**3x2**	0.050	3x3	0.03	4x2
dc1_02*	0.119	3x4	**0.029**	**3x4**	0.080	4x3	0.037	3x4	0.091	3x5	N/A	-
dc1_03	0.203	4x3	**0.070**	**4x3**	0.187	4x3	0.070	4x3	0.070	4x5	N/A	-
dc1_04	0.066	2x3	0.025	2x3	0.064	2x3	**0.023**	**2x3**	0.056	2x4	N/A	-
ex5_03	0.003	7x1	**0.002**	**7x1**	0.003	7x1	0.004	7x1	0.199	7x1	N/A	-
ex5_04	0.003	8x1	0.003	8x1	0.003	8x1	**0.002**	**8x1**	0.113	8x1	N/A	-
ex5_05	0.003	6x1	0.003	6x1	0.003	6x1	**0.002**	**6x1**	0.058	6x1	N/A	-
ex5_06	2.777	3x6	5.814	3x6	3.575	3x6	0.955	**3x6**	**0.477**	3x10	N/A	-
ex5_07	167.767	4x6	**	**	578.060	4x6	26.843	**4x6**	**0.288**	3x13	N/A	-
ex5_08	11.254	3x7	325.598	3x7	50.687	3x7	**0.004**	**3x7**	1.389	3x9	N/A	-
ex5_09	9.757	4x6	**	**	261.837	4x6	6.642	**4x6**	**5.424**	3x11	N/A	-
ex5_10	1.463	3x6	3.994	3x6	1.828	3x6	**0.39**	**3x6**	0.178	3x9	N/A	-
ex5_11	0.003	2x8	**	**	0.004	2x8	**0.002**	**2x8**	0.997	2x10	N/A	-
ex5_12*	6.026	3x6	1.450	3x5	8.097	3x5	**0.208**	**3x5**	2.969	5x9	N/A	-
ex5_13	41.045	4x6	**	**	231.295	4x6	11.494	**3x8**	**0.720**	3x13	N/A	-
ex5_14	3.751	2x8	211.941	2x8	4.091	2x8	0.368	**2x8**	**0.231**	3x11	N/A	-
ex5_15	**	**	**	**	**	**	59.780	**4x7**	**1.658**	4x13	N/A	-
ex5_16	**0.002**	**2x5**	0.003	2x5	0.003	2x5	0.004	2x5	0.108	2x7	N/A	-
ex5_17	**	**	**	**	**	**	8483.315	**4x7**	**121.374**	4x10	N/A	-
ex5_18	0.003	2x7	**0.002**	**2x7**	0.024	2x7	0.005	2x7	0.214	2x9	N/A	-
ex5_19	3.962	3x6	8.337	3x6	4.109	3x6	**0.286**	**3x6**	1.238	5x7	N/A	-
ex5_20	0.003	2x6	**0.002**	**2x6**	0.003	2x6	0.003	2x6	0.332	3x8	N/A	-
ex5_21	287.766	3x7	185.592	3x7	12.478	3x7	**0.956**	**3x7**	1.368	4x9	N/A	-
ex5_22	3.002	3x6	9.851	3x6	4.303	3x6	0.146	**3x6**	**0.031**	3x8	N/A	-
ex5_23	**	**	**	**	**	**	10598.277	**4x8**	**0.116**	4x11	N/A	-
ex5_24	**	**	**	**	**	**	698.092	**5x6**	**0.163**	5x14	N/A	-
ex5_25	14.172	3x7	513.478	3x7	56.508	3x7	**0.386**	**3x7**	0.740	3x8	N/A	-
ex5_26	108.275	**3x7**	**	**	167.706	3x7	15.257	3x7	**1.368**	4x11	N/A	-
ex5_27*	1779.261	3x8	**	**	1348.150	4x6	**21.092**	**4x6**	**1.130**	4x10	N/A	-
ex5_28*	25.564	4x6	**	**	51.239	6x4	1.374	**3x8**	**1.232**	3x13	N/A	-
misex1_00	0.087	4x2	0.038	4x2	0.083	4x2	**0.024**	**4x2**	0.040	4x3	0.08	2x4
misex1_01	1.872	3x5	0.516	3x5	1.981	3x5	**0.242**	**3x5**	0.401	5x5	N/A	-
misex1_02	15.966	5x4	425.897	5x4	26.032	5x4	14.289	**5x4**	**0.702**	5x5	N/A	-
misex1_03	1.574	4x3	0.235	**4x3**	1.413	4x3	0.361	4x3	**0.130**	4x6	0.53	6x4
misex1_04	0.266	**3x4**	0.085	**3x4**	0.227	**3x4**	0.231	3x5	**0.0762**	4x7	N/A	-
misex1_05	3.211	4x4	6.360	4x4	3.870	4x4	0.966	**4x4**	**0.345**	4x6	N/A	-
misex1_06	1.854	5x3	2.354	3x5	1.689	5x3	0.799	**5x3**	**0.124**	4x7	N/A	-
misex1_07	0.667	4x3	0.167	**4x3**	0.601	4x3	0.208	4x3	**0.062**	5x5	N/A	-
mp2d_00	0.003	1x11	**0.002**	**1x11**	0.004	1x11	0.003	1x11	1.250	2x13	N/A	-
mp2d_01	**	**	**	**	**	**	68.428	**5x7**	**0.514**	4x11	N/A	-
mp2d_02	**	**	**	**	**	**	22.846	**4x9**	**0.395**	4x13	N/A	-
mp2d_03*	1241.821	**4x6**	**	**	2603.411	6x4	191.469	5x5	**0.545**	7x6	N/A	-
mp2d_04	1816.681	7x3	**	**	3512.153	7x3	23.751	**7x3**	24.020	7x3	**3.68**	6x5
mp2d_05	0.003	5x1	**0.002**	**5x1**	0.003	5x1	0.003	5x1	0.539	5x1	N/A	-
mp2d_06	0.397	4x3	0.161	6x2	0.395	4x3	**0.103**	**6x2**	220.463	5x4	0.14	5x5
mp2d_07	0.003	8x1	**0.002**	**8x1**	0.009	8x1	0.003	8x1	0.034	8x1	N/A	-
mp2d_08	0.003	1x5	**0.002**	**1x5**	0.003	1x5	0.003	1x5	0.034	2x7	N/A	-
newapla2_00	0.003	6x1	**0.002**	**6x1**	0.003	6x1	0.003	6x1	0.023	6x1	N/A	-
newbyte_00	0.003	5x1	**0.002**	**5x1**	0.004	5x1	0.003	5x1	0.064	5x1	N/A	-
newtag_00	6.103	3x6	10.921	3x6	7.719	3x6	**0.262**	**3x6**	0.842	3x8	N/A	-

+ Bold values represent the best results; "*" indicates the non-optimal solutions of Algorithm in [17] that are different than the fixed version; "**" indicates time-out; "N/A" is used for non-D-reducible functions.

Table 2: Comparison of Optimization Algorithms

	Fixed Version of [17] (Optimal)		**Proposed** Heuristic (non-Optimal)		P-Decomposition (non-Optimal) [14]		D-Reducing (non-Optimal) [15]	
Benchmark Name	Elapsed Time (s)	Size	Elapsed Time (s)	Size	Elapsed Time (s)	Size	Elapsed Time (s)	Size
5xp1_00	21.867	**5x4**	**2.305**	5x5	3.599	5x8	N/A	-
5xp1_01	**	**	1405.815	**5x5**	**1.697**	5x10	N/A	-
5xp1_03	**	**	1062.801	4x15	**10.716**	**4x11**	N/A	-
5xp1_04	15.812	4x3	**0.020**	**4x3**	0.609	5x10	N/A	-
5xp1_05	0.285	3x4	**0.209**	**3x4**	0.065	4x6	N/A	-
5xp1_06	0.003	3x3	**0.002**	**3x3**	0.031	3x3	N/A	-
5xp1_07	0.002	2x2	**0.002**	**2x2**	0.015	2x2	0.03	2x2
5xp1_08	**0.002**	**1x1**	0.002	1x1	0.017	1x1	N/A	-
5xp1_09	1.179	4x3	0.039	**4x3**	0.237	4x4	**0.06**	5x3
bw_00	1.147	5x3	0.341	**5x3**	**0.069**	4x6	N/A	-
bw_01	**0.002**	**3x3**	0.003	3x3	0.04	3x4	0.05	5x2
bw_02	0.33	4x3	**0.046**	**4x3**	0.062	3x5	0.06	5x3
bw_03	0.248	3x4	**0.020**	**3x4**	0.084	3x6	N/A	-
bw_04	0.412	4x3	**0.049**	**4x3**	0.212	5x5	N/A	-
bw_05	0.513	5x3	**0.04**	**5x3**	0.069	3x7	N/A	-
bw_06	0.657	4x3	**0.165**	**4x3**	0.283	5x6	N/A	-
bw_07	0.134	4x3	0.071	**4x3**	0.11	4x5	**0.06**	5x4
bw_08	0.766	**3x4**	0.319	5x4	**0.121**	3x6	N/A	-
bw_09	**0.002**	**3x3**	0.007	3x3	0.056	3x4	N/A	-
bw_10	0.036	4x2	**0.003**	**4x2**	0.143	5x2	0.07	2x4
bw_11	0.358	4x3	0.04	**4x3**	0.058	3x5	**0.15**	5x3
bw_12	0.002	3x3	**0.005**	**3x3**	0.051	3x4	N/A	-
bw_13	0.42	4x3	0.087	**4x3**	**0.068**	4x5	N/A	-
bw_14	0.357	5x2	**0.064**	**5x2**	0.066	3x4	0.16	5x5
bw_15	0.236	4x3	0.059	**4x4**	0.279	4x4	**0.13**	5x4
bw_16	**0.002**	**3x3**	0.002	3x3	0.056	3x4	N/A	-
bw_17	1.463	**3x5**	0.215	4x4	0.38	4x7	**0.16**	6x3
bw_18	0.368	**3x4**	**0.066**	4x4	0.078	4x5	0.17	6x3
bw_19	1.005	3x5	0.112	**3x5**	**0.104**	3x6	N/A	-
bw_20	0.223	3x4	**0.015**	**3x4**	0.231	4x5	N/A	-
bw_21	**0.002**	**5x1**	0.003	5x1	0.028	5x1	N/A	-
bw_22	1.392	**3x5**	**0.072**	4x4	0.364	4x6	N/A	-
bw_23	1.099	**5x3**	**0.163**	4x4	0.504	5x6	N/A	-
bw_24	0.261	2x5	**0.094**	**2x5**	0.112	2x6	N/A	-
bw_25	1.258	**3x5**	**0.082**	4x4	0.172	4x7	N/A	-
bw_26	0.898	**3x5**	0.135	5x4	**0.078**	3x6	N/A	-
bw_27	**0.002**	**5x1**	0.003	5x1	0.071	5x1	N/A	-
inc_00	8.420	**4x4**	1.557	5x4	**0.451**	5x7	N/A	-
inc_01	17.425	**5x4**	12.512	5x5	**0.506**	5x7	N/A	-
inc_02	**	**	6020.116	**4x11**	**5.227**	5x10	N/A	-
inc_04	7.395	**4x4**	0.833	5x4	**0.033**	6x8	N/A	-
inc_05	0.567	5x2	0.107	**5x2**	**0.03**	4x3	0.04	5x3
inc_06	1.277	4x3	**0.003**	**4x3**	0.02	4x3	N/A	-
inc_07	0.994	**4x3**	0.072	5x3	0.18	5x4	**0.16**	5x3
inc_08	0.003	3x2	**0.003**	**3x2**	0.021	3x2	0.03	4x2
misex3c_00	1.145	3x5	**0.073**	**3x5**	**	**	N/A	-
misex3c_01	3.908	**3x5**	**0.072**	4x5	**	**	N/A	-
misex3c_02	0.065	4x3	**0.024**	**4x3**	36.467	5x10	N/A	-
misex3c_03	165.451	**4x5**	**16.155**	3x8	259.354	4x7	N/A	-
misex3c_04	262.09	**3x7**	**9.294**	4x6	164.132	5x5	N/A	-
misex3c_06	**	**	1563.699	5x6	**3.996**	**4x5**	N/A	-
rd73_02	**	**	81.927	35x4	**4.195**	**5x16**	N/A	-
t481	**	**	**91.894**	**9x8**	**	**	N/A	-
vg2_00	**	**	**68.74**	**9x4**	**	**	N/A	-
vg2_02	**	**	**3.509**	**9x4**	**	**	N/A	-
vg2_05	22.738	**4x4**	1.857	4x5	**0.5**	4x5	N/A	-
vg2_07	16.067	4x4	**1.679**	4x5	15.782	**4x4**	N/A	-

+ Bold values represent the best results; "**" indicates time-out; "N/A" is used for non-D-reducible functions.