Improved Heuristics for Low-latency Implementations of Linear Layers (Full Version)

Qun Liu[1,2], Zheng Zhao[1,2], and Meiqin Wang(✉)[1,2,3]

[1] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, China, qunliu@mail.sdu.edu.cn
[2] School of Cyber Science and Technology, Shandong University, Qingdao, China, zhaozheng@mail.sdu.edu.cn
[3] Quan Cheng Shandong Laboratory, Jinan, China, mqwang@sdu.edu.cn

Abstract. In many applications, low area and low latency are required for the chip-level implementation of cryptographic primitives. The low-cost implementations of linear layers usually play a crucial role for symmetric ciphers. Some heuristic methods, such as the forward search and the backward search, minimize the number of XOR gates of the linear layer under the minimum latency limitation.
For the sake of achieving further optimization for such implementation of the linear layer, we put forward a new general search framework attaching the *division optimization* and *extending base* techniques in this paper. In terms of the number of XOR gates and the searching time, our new search algorithm is better than the previous heuristics, including the forward search and the backward search when testing matrices provided by them. We obtain an improved implementation of AES MixColumns requiring only 102 XORs under minimum latency, which outdoes the previous best record provided by the forward search.

Keywords: Lightweight cryptography · Linear layers · Low latency · AES

1 Introduction

In recent years, lightweight cryptography has been applied to provide security and privacy in many fields, such as Internet of Things (IoTs), wireless sensor networks, and Radio-Frequency IDentification (RFID) tags. These devices limit the use of resources, such as circuit size, power consumption, and latency. Various restrictions may lead to new security threats in design, cryptanalysis, and implementation. Generally, lightweight cryptography ensures secure encryption and satisfies the requirement of limited resource.

Research on lightweight cryptography usually falls in two directions. The first direction focuses on designing new ciphers based on lightweight components. They are efficient in either hardware or software implementations. Plenty of related works have been introduced [5,8,20]. Another direction is to optimize the implementation of existing ciphers. The second direction often boils down to

the optimization of lightweight diffusion and confusion components. The Sbox is one of the most popular confusion components of symmetric-key ciphers. Many tools are proposed to optimize the primitive, such as `LIGHTER` [17] and `PEIGEN` [7]. In addition, the diffusion components are essential matrices, and the Maximal Distance Separable (MDS) matrices are the most well-known diffusion components.

In terms of implementation for lightweight cryptography primitives, there are many criteria. The most popular one should be the gate equivalents (GE) required by the chip-level implementation of the cryptographic algorithm.[4] GE effectively approximates the complexity of digital electronic circuits. Generally, two components are relevant to the cost. The diffusion component, i.e., the linear layer, is typically realized with many `XOR` gates. Reducing `XOR` operations will lead to a non-negligible decreasing the number of GE. Such optimization for the number of `XOR`s can be formulated as the Shortest Linear Program **(SLP)** problem. Although it has been shown to be an NP-hard problem [11], there is still a growing body of work solely concentrating on decreasing the GE. More and more concerns for heuristics searching for sub-optimal solutions have arisen (see [4,12,19,30,31] for an incomplete list).

Therefore, as an important criterion, latency has been attracting more and more attention. Many of the applications require low latency, including automobiles, robots, or mission-critical computation applications. It impacts the throughput of encryption/decryption and plays an important role in the low-energy consideration of ciphers [5]. In CHES 2021, Leander *et al.* propose a new cipher `SPEEDY`, which explores a low-latency architecture. Usually, the depth of the circuit can be utilized to approximate the latency. The depth is the critical path length of the circuit. The low-latency optimization for linear layers is formulated as the Shortest Linear Program problem with the minimum Depth **(SLPD)**.

Focusing on optimizing low-latency implementations, there are two kinds of heuristics. The first one is the *forward search* and the second one is the *backward search*. For the forward search algorithm, firstly, Li *et al.* provided a method by adding a depth constraint in BP algorithm (called LSL algorithm) [21], and BP algorithm is given in [12]. Subsequently, the LSL algorithm was adapted by Banik *et al.* [6] by considering the influence of different permutations for matrices. The backward search algorithm is constructed by Liu *et al.* in [25].

1.1 Our Contributions

For the sake of achieving further optimization for linear matrices, the new heuristics for them is important. This paper focuses on improving the previous heuristics for the low-latency implementations of linear layers. For the forward search and backward search, we find that many good candidate implementations have

[4] The unit of gate size is Gate Equivalent (GE), where one GE equals the area of a 2-input `NAND` gate. The cost of other gates in terms of GE is a normalized ratio between their area and one `NAND` gate area.

been discarded because the search space has been reduced greatly. Therefore, we aim to deal with this problem and propose a new search framework attaching two optimization techniques.

The framework splits the given circuit and extends the base to optimize the parallel circuit step by step to finish the whole circuit optimization for any heuristics. We notice that the candidate circuits recommended by the heuristics are strongly dependent on all the output signals. The fewer optimized output signals may lead to better circuits. We provide the *division optimization technique* by splitting the output signals and step-wise optimizing, which can provide more good candidate implementations. In addition, different heuristics usually utilize different search spaces. Some good circuits will never be recommended. Thus, we put forward the *extending base technique* to break through the limitation of the forward search and backward search.

Based on the above techniques, we propose a general optimization aiming at improving the given circuit for matrices. Concretely, rather than optimizing the complete matrices, the framework only takes a partial circuit into account and extends additional base values for the heuristics.

We apply the framework to linear layers of block ciphers and find many low-latency candidates for implementation. The benchmark results are shown in Table 1. Although these matrices have been optimized by the forward search and backward search, we still improved 9 of them. One particularly interesting case is that we obtain an implementation of AES MixColumns requiring only 102 XORs with depth 3, which breaks the previous record with 103 XORs. We also apply the framework to 4254 MDS matrices proposed in [21], and achieve better implementations in XOR gates for 77.5% of them. From these matrices, we find a smaller matrix requiring 85 XOR gates (reducing one gate than before).

Then, we synthesize the results of AES MixColumns using the ASIC library named UMC 55 nm, which shows that our implementation has lower power and latency.

1.2 Organization

In Section 2, we give some basic notations and metrics. Moreover, in Section 3, we discuss the problems of existing heuristics and propose two optimization techniques. The general optimization framework is introduced in Section 4. All the results and implementations in hardware are given in Section 5. Finally, we conclude and propose future research directions in Section 6.

2 Preliminaries

2.1 Notations

Let $\mathbb{F}_2$ be the finite field with two elements 0 and 1 and $\mathbb{F}_2^n$ be the the vector space of all n-dimensional vectors over $\mathbb{F}_2$. $M_{m\times n}$ denotes an $m \times n$ matrix over $\mathbb{F}_2$ and $wt(M)$ denotes the Hamming weight of a matrix M over $M_{m\times n}$, which counts the number of 1's contained in M.

Table 1: The XOR number/depth of implementation costs of matrices.

Matrix	Size	[19]	[31]	[23]	[21]*	[6]*	[25]*	**This paper***
AES [14]	32	97/8	92/6	91/7	105/3	103/3	103/3	**102/3**
SMALLSCALE AES [13]	16	47/7	43/5	43/5	49/3	49/3	47/3	47/3
JOLTIK [16]	16	48/4	44/7	43/8	51/3	50/3	48/3	48/3
QARMA128 [3]	32	48/3	48/3	48/3	48/2	48/2	48/2	48/2
MIDORI [5]	16	24/4	24/3	24/3	24/2	24/2	24/2	24/2
PRINCE M_0,M_1 [10]	16	24/4	24/6	24/6	24/2	24/2	24/2	24/2
PRIDE $L_0 - L_3$ [1]	16	24/3	24/3	24/3	24/2	24/2	24/2	24/2
QARMA64 [3]	16	24/3	24/5	24/5	24/2	24/2	24/2	24/2
SKINNY64 [8]	16	12/2	12/2	12/2	12/2	12/2	12/2	12/2
CAMELLIA [2]	8	16/4	16/4	16/4	20/3	-	19/3	19/3
[19]	32	84/4	-	-	96/3	-	92/3	**89/3**
[29](Hadamard)	16	48/3	44/7	44/7	51/3	50/3	49/3	**48/3**
[24](Circulant)	16	44/3	44/6	43/4	47/3	44/3	44/3	44/3
[22](Circulant)	16	44/5	44/8	43/4	47/3	44/3	44/3	44/3
[9](Circulant)	16	42/5	41/6	40/5	47/3	43/3	45/3	43/3
[28](Toeplitz)	16	43/5	41/7	40/7	44/3	43/3	45/3	43/3
[17]	16	43/5	41/6	40/6	45/3	45/3	45/3	**44/3**
[29](Involutory)	16	48/4	44/8	43/8	51/3	49/3	48/3	48/3
[22](Involutory)	16	48/4	44/6	43/8	51/3	49/3	48/3	48/3
[28](Involutory)	16	42/4	38/8	37/7	48/3	46/3	45/3	**43/3**
[17](Involutory)	16	47/7	41/6	41/10	47/3	47/3	47/3	47/3
[29](Hadamard)	32	100/5	90/6	91/7	102/3	99/3	100/3	99/3
[24](Circulant)	32	112/5	121/11	107/6	114/3	113/3	113/3	**112/3**
[22]	32	102/3	104/6	99/4	102/3	103/3	102/3	102/3
[9](Circulant)	32	110/5	114/10	105/7	112/3	110/3	111/3	110/3
[28](Toeplitz)	32	107/5	114/12	100/9	107/3	107/3	107/3	107/3
[17](Subfield)	32	86/5	82/7	80/6	90/3	90/3	93/3	90/3
[29](Involutory)	32	100/6	91/6	89/8	102/3	100/3	100/3	**99/3**
[22](Involutory)	32	91/6	87/6	86/9	99/3	95/3	94/3	**93/3**
[28](Involutory)	32	100/6	93/8	92/8	104/4	102/4	109/4	102/4
[17](Involutory)	32	91/7	83/6	84/6	94/3	94/3	97/3	94/3
[21](Involutory)	32	-	-	-	88/3	-	86/3	**85/3**

* The results take the number of XOR gates into account with respect to the minimum depth.

Given the matrix M and the input values $\vec{t} = (t_0, t_1, ..., t_{n-1})^T$, each output value y_i can be computed by $a_{i0}t_0 \oplus a_{i1}t_1 \oplus \ldots \oplus a_{i(n-1)}t_{n-1}$, where each coefficient a_{ij} is the entry of matrix M at i-th row and j-th column. We can then associate y_i with a binary vector:

$$[a_{i0}, a_{i1}, \ldots, a_{i(n-1)}]. \tag{1}$$

Generally, every value t can be computed by $a_0t_0 \oplus a_1t_1 \oplus \ldots \oplus a_{n-1}t_{n-1}$ and is associated with $[a_0, a_1, \ldots, a_{n-1}]$.

For three values t_1, t_2, and t_3, we say t_2 and t_3 generate t_1 if $t_1 = t_2 \oplus t_3$ with $\oplus$ element-wise plus is included in the circuit. We define its depth $\mathcal{D}(t)$ as the maximum number of `XOR` gates of a path from input values to t. For each input value t_i, $\mathcal{D}(t_i)$ is 0. The depth of a circuit is the critical path length of the circuit. For each value t, the minimum depth $\mathcal{D}_{min}(t)$ is defined as

$$\lceil \log_2(wt(t)) \rceil. \tag{2}$$

Suppose that a set A contains different values, the depth of A is defined as

$$\mathcal{D}(A) = \max_{v \in A}\{\mathcal{D}(v)\}, \tag{3}$$

and the minimum depth of A is defined as

$$\mathcal{D}_{min}(A) = \max_{v \in A}\{\lceil \log_2(wt(v)) \rceil\}. \tag{4}$$

Similarly, the minimum depth of a matrix M is defined as

$$\mathcal{D}_{min}(M) = \max_{y_i \in M}\{\lceil \log_2(wt(y_i)) \rceil\}, \tag{5}$$

where y_i is the i-th output value of M. Finding a circuit with respect to the minimum depth means that the depth of the circuit equals the minimum depth of M. For the circuit $\mathcal{C}$, we also use $\mathcal{D}(\mathcal{C})$ to represent the depth of $\mathcal{C}$.

2.2 SLP Problem and SLPD

Definition 1 ([11]). *The Shortest Linear Program (SLP) problem is defined as finding a solution with the least `XOR` gates to compute M over $M_{m \times n}$.*

The problem is extended by considering the depth of the solution [6,21,25]. We call it the SLP problem with respect to the minimum Depth (SLPD). The solution always reaches the minimum depth with the smallest `XOR` gates.

A possible solution is the exhaustive search method, which is discussed in Supplementary Materials A. Unfortunately, most of matrices used in linear layers are too larger to utilize the exhaustive search. Thus, different heuristics are used to optimize the matrices.

2.3 State-of-the-art Works

Two heuristics to solve SLPD have been presented, which are the forward search and the backward search, respectively.

Forward search. Forward search is based on the BP algorithm [12], which combines input signals to reach the output signals. We review the algorithm in the following (see Algorithm 1).

Given a binary matrix M, the input signals are $\{t_0, t_1, ..., t_{n-1}\}$ and the output signals are $\{y_0, y_1, ..., y_{m-1}\}$. The base set $\mathcal{B}$ and the output set $\mathcal{O}$ contain all the input signals and output signals, respectively. Then, they initialize an m-integer vector $Dist$ which keeps track of the distances of each target value from $\mathcal{B}$. The $Dist$ is $[\delta(\mathcal{B}, y_0), \delta(\mathcal{B}, y_1), ..., \delta(\mathcal{B}, y_{m-1})]$, where $\delta(\mathcal{B}, y_i)$ indicates the minimum number of XOR gates required that can obtain y_i from $\mathcal{B}$. Then, they repeatedly pick two values from $\mathcal{B}$, add them together as a new value, and puts the new value into $\mathcal{B}$. Such update process is based on the following rules.

- **Rule 1:** Perform `XOR` on every unique pair of values in $\mathcal{B}$ to generate a new value. The new value is used to re-evaluate the $Dist$ vector, and calculate the new distance $\sum_{i=0}^{m-1} Dist[i]$.
- **Rule 2:** If a pair can generate the target signal, then choose it first. Otherwise, select the smallest $\sum_{i=0}^{m-1} Dist[i]$ and put the corresponding value into $\mathcal{B}$. In case of tie, use the Euclidean norm of $Dist$ to determine which candidate is better.
- **Rule 3:** If there still exist many candidates, choose one randomly.

Algorithm 1 BP Algorithm

Input: A matrix M over $M_{m\times n}$
Output: A circuit $\mathcal{C}$ to implement M
1: Initial the base set $\mathcal{B} \leftarrow \{t_0, t_1, \ldots, t_{n-1}\}$
2: Initial the output set $\mathcal{O} \leftarrow \{y_0, y_1, \ldots, y_{m-1}\}$
3: Initial the circuit $\mathcal{C} \leftarrow \phi$
4: **while** $\mathcal{O} \neq \phi$ **do**
5: Choose a candidate $t_{|\mathcal{B}|} = t_i \oplus t_j$ based on Rule 1, Rule 2, and Rule 3
6: **if** $t_{|\mathcal{B}|} \in \mathcal{O}$ **then**
7: $\mathcal{O} \leftarrow \mathcal{O}/\{t_{|\mathcal{B}|}\}$
8: **end if**
9: $\mathcal{C} \leftarrow \mathcal{C} \cup \{t_{|\mathcal{B}|} = t_i \oplus t_j\}$
10: $\mathcal{B} \leftarrow \mathcal{B} \cup \{t_{|\mathcal{B}|}\}$
11: **end while**
12: **return** $\mathcal{C}$

Because the original BP Algorithm is not applicable in low-latency scenario, Li *et al.* [21] enhance the algorithm with circuit depth awareness (called LSL algorithm). Overall, they append a function $Pick()$ to choose two values from

the base set $\mathcal{B}$ to generate new value, in which the depth of the new value can not exceed a specified depth bound. Other steps of the algorithm is the same as the BP algorithm.

In order to improve the LSL algorithm, Banik *et al.* [6] modify the target matrix M by adding permutations. Specifically, they generate two permutations P and Q and let $M_R = P \cdot M \cdot Q$. A permutation only shuffles the rows and columns of M and keeps the linear relation unchanged. Then, they run the LSL algorithm many times for different M_R to find better circuits. Through their idea, additional randomness can be introduced in the original matrix.

We take an example to show the forward search. Suppose that the target matrix M_1 is

$$\begin{bmatrix} 1&1&0&0&0&0 \\ 1&1&1&0&0&0 \\ 1&1&1&1&0&0 \\ 1&1&1&1&1&0 \\ 1&1&1&1&0&1 \end{bmatrix}.$$

The input set $\mathcal{B}$ is $\{t_0, t_1, t_2, t_3, t_4, t_5\}$ and the output set $\mathcal{O}$ is $\{y_0, y_1, y_2, y_3, y_4\}$. Then, let the circuit satisfy the low-latency limitation $\mathcal{L} = \{\mathcal{D}(M_1) = \mathcal{D}_{min}(M_1) = 3\}$. The initial $Dist$ is $[1, 2, 3, 4, 4]$. In Table 2, we provide the circuit produced by the forward search. Note that the depth of t_8 is 3, and then t_8 can not be used in the subsequent optimization. t_9 and t_{11} are generated to meet the limitation.

Table 2: The implementation of M_1 using the forward search.

No.	**Operation**	**Depth**	**New value**	**New dist**
1	$t_6 = t_0 \oplus t_1 // y_0$	1	$t_6 = [1,1,0,0,0,0]$	$[\mathbf{0},\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{3}] = 9$
2	$t_7 = t_6 \oplus t_2 // y_1$	2	$t_7 = [1,1,1,0,0,0]$	$[0,\mathbf{0},\mathbf{1},\mathbf{2},\mathbf{2}] = 5$
3	$t_8 = t_7 \oplus t_3 // y_2$	3	$t_8 = [1,1,1,1,0,0]$	$[0,0,\mathbf{0},\mathbf{1},\mathbf{1}] = 2$
4	$t_9 = t_3 \oplus t_4$	1	$t_9 = [0,0,0,1,1,0]$	$[0,0,0,1,1] = 2$
5	$t_{10} = t_7 \oplus t_9 // y_3$	3	$t_{10} = [1,1,1,1,1,0]$	$[0,0,0,\mathbf{0},1] = 1$
6	$t_{11} = t_3 \oplus t_5$	1	$t_{11} = [0,0,0,1,0,1]$	$[0,0,0,0,1] = 1$
7	$t_{12} = t_7 \oplus t_{11} // y_4$	3	$t_{12} = [1,1,1,1,0,1]$	$[0,0,0,0,\mathbf{0}] = 0$

Backward search. The backward search is proposed in [25], which iteratively splits the output values until all the input values appear. The backward search utilizes a completely different strategy with the low-latency metric. In the algorithm, the output values and input values are put into the working set $\mathcal{W}$ and the input set $\mathcal{X}$, respectively. Then, the predecessor set $\mathcal{P}$ saves the values that can be used to split $\mathcal{W}$.

To exemplify this algorithm, we give an example to show the backward search. Suppose that the target matrix M_2 is

$$\begin{bmatrix} 1\,1\,1\,1\,1\,1\,0\,0 \\ 0\,0\,0\,0\,1\,1\,1\,1 \\ 0\,0\,0\,0\,0\,1\,1\,1 \\ 0\,0\,0\,0\,0\,1\,1\,0 \end{bmatrix}.$$

The input signals are $t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7$ and the target signals are y_0, y_1, y_2, y_3.

- **Initialization.** In M_2, y_i represents the i-th row of the matrix, and t_j can be represented by the unit vector with the j-th bit 1. We set the predecessor set $\mathcal{P} = \phi$, the working set $\mathcal{W} = \{y_0, y_1, y_2, y_3\}$, and $\mathcal{X} = \{t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$. Then, we have

$$\mathcal{D}_{min}(y_0) = 3, \mathcal{D}_{min}(y_1) = 2, \mathcal{D}_{min}(y_2) = 2, \mathcal{D}_{min}(y_3) = 1.$$

- **Step 1**. $D_{min}(\mathcal{W}) = 3$. If a value $t \in \mathcal{W}$ and $\mathcal{D}_{min}(t) < 3$, we will put it into $\mathcal{P}$ from $\mathcal{W}$. Therefore, $\mathcal{W} = \{y_0\}$, and $\mathcal{P} = \{y_1, y_2, y_3\}$.
- **Step 2**. Generate $t_8 = [1, 1, 1, 1, 0, 0, 0, 0]$ and $t_9 = [0, 0, 0, 0, 1, 1, 0, 0]$ to split y_0 by $y_0 = t_8 \oplus t_9$. Therefore, $\mathcal{W} = \phi$. $\mathcal{P} = \{y_1, y_2, y_3, t_8, t_9\}$. Since $\mathcal{W} = \phi$, let $\mathcal{W} = \mathcal{P}$ and $\mathcal{P} = \phi$. Now $D_{min}(\mathcal{W}) = 2$. Then, we put y_3 and t_9 into $\mathcal{P}$.
- **Step 3**. Produce $t_{10} = [0, 0, 0, 0, 0, 0, 1, 1]$ to split y_1 by $y_1 = t_9 \oplus t_{10}$. Now, $\mathcal{W} = \{y_2, t_8\}$, and $\mathcal{P} = \{y_3, t_9, t_{10}\}$.
- **Step 4**. Split y_2 by $y_2 = t_5 \oplus t_{10}$. Now $\mathcal{W} = \{t_8\}$ and $\mathcal{P} = \{y_3, t_5, t_9, t_{10}\}$. Create $t_{11} = [1, 1, 0, 0, 0, 0, 0, 0]$ and $t_{12} = [0, 0, 1, 1, 0, 0, 0, 0]$ to split t_8. Then, $\mathcal{W} = \phi$, $\mathcal{P} = \{y_3, t_5, t_9, t_{10}, t_{11}, t_{12}\}$.
- **Step 5**. Since $\mathcal{W} = \phi$, let $\mathcal{W} = \mathcal{P}$ and $\mathcal{P} = \phi$. The maximum depth is $D_{min}(\mathcal{W}) = 1$. y_3, t_9, t_{10}, t_{11} and t_{12} can be split by the unit vectors. We show the complete circuit in Table 3.

Table 3: The splitting process of M_2 using the backward search.

No.	Operation	Depth	New value	Minimum depth
1	$y_0 = t_8 \oplus t_9$	3	$y_0 = [1, 1, 1, 1, 1, 1, 0, 0]$	$\mathcal{D}_{min}(y_0) = 3$
2	$y_1 = t_9 \oplus t_{10}$	2	$y_1 = [0, 0, 0, 0, 1, 1, 1, 1]$	$\mathcal{D}_{min}(y_1) = 2$
3	$y_2 = t_5 \oplus t_{10}$	2	$y_2 = [0, 0, 0, 0, 0, 1, 1, 1]$	$\mathcal{D}_{min}(y_2) = 2$
4	$t_8 = t_{11} \oplus t_{12}$	2	$t_8 = [1, 1, 1, 1, 0, 0, 0, 0]$	$\mathcal{D}_{min}(t_8) = 2$
5	$t_{10} = t_6 \oplus t_7$	1	$t_{10} = [0, 0, 0, 0, 0, 0, 1, 1]$	$\mathcal{D}_{min}(t_{10}) = 1$
6	$t_{11} = t_0 \oplus t_1$	1	$t_{11} = [1, 1, 0, 0, 0, 0, 0, 0]$	$\mathcal{D}_{min}(t_{11}) = 1$
7	$t_{12} = t_2 \oplus t_3$	1	$t_{12} = [0, 0, 1, 1, 0, 0, 0, 0]$	$\mathcal{D}_{min}(t_{12}) = 1$
8	$t_9 = t_4 \oplus t_5$	1	$t_9 = [0, 0, 0, 0, 1, 1, 0, 0]$	$\mathcal{D}_{min}(t_9) = 1$
9	$y_3 = t_5 \oplus t_6$	1	$y_3 = [0, 0, 0, 0, 0, 1, 1, 0]$	$\mathcal{D}_{min}(y_3) = 1$

The complete algorithm for the backward search can be seen in Algorithm 2. $\mathcal{W}$ and $\mathcal{P}$ are matched $\mathcal{O}$ and $\mathcal{B}$ in the forward search, respectively. The difference is that both $\mathcal{W}$ and $\mathcal{P}$ are dynamically changed. $\mathcal{X} \cup \mathcal{W} \neq \mathcal{X}$ indicates that there is at least one non-input value in $\mathcal{W}$, which will be split according to the following five rules, which are used to reduce the search space.

- **Rule 1:** If $\mathcal{D}_{min}(t) < \mathcal{D}_{min}(\mathcal{W})$ $(t \in \mathcal{W})$, t will be put into $\mathcal{P}$ (see Step 1 in the example of M_2).
- **Rule 2:** If $\exists p_1, p_2 \in \mathcal{P}$, $\exists w \in \mathcal{W}$ s.t. $w = p_1 \oplus p_2$, w will be removed from $\mathcal{W}$.
- **Rule 3:** If $\exists p_1 \in \mathcal{P}$, $\exists w \in \mathcal{W}$ s.t. $p_2 = w \oplus p_1$ and $\mathcal{D}_{min}(p_2) < \mathcal{D}_{min}(w)$, remove w and append p_2 in $\mathcal{P}$ (see Step 3 in the example of M_2).
- **Rule 4:** If $\exists p_1, p_2, p_3$, $\exists w_1, w_2 \in \mathcal{W}$ s.t. $w_1 = p_1 \oplus p_2, w_2 = p_2 \oplus p_3$, where $\mathcal{D}_{min}(p_1), \mathcal{D}_{min}(p_2), \mathcal{D}_{min}(p_3) < \max(\mathcal{D}_{min}(w_1), \mathcal{D}_{min}(w_2))$, remove w_1 and w_2, and put p_1, p_2 and p_3 into $\mathcal{P}$.
- **Rule 5:** This is the default rule. Split w $(w \in \mathcal{W})$ into p_1 and p_2 $(\mathcal{D}_{min}(p_1) < \mathcal{D}_{min}(w), \mathcal{D}_{min}(p_2) < \mathcal{D}_{min}(w))$. p_1 and p_2 are put into $\mathcal{P}$ (see Step 2 in the example of M_2).

Algorithm 2 Backward Search

Input: A matrix M over $M_{m \times n}$
Output: A circuit $\mathcal{C}$ to implement M

$\mathcal{W} \leftarrow \{y_0, y_1, \ldots, y_{m-1}\}$ ▷ the working set
$\mathcal{X} \leftarrow \{t_0, t_1, \ldots, t_{n-1}\}$ ▷ the input set
$\mathcal{P} \leftarrow \phi$ ▷ the predecessor set
$\mathcal{C} \leftarrow \phi$ ▷ the circuit
while $\mathcal{X} \cup \mathcal{W} \neq \mathcal{X}$ **do**
    while $\mathcal{W} \neq \phi$ **do**
        choose a value $w \in \mathcal{W}$ by Rule 1-Rule 5 and split w by $w = p \oplus q$.
        if $p \notin \mathcal{P}$ **then**
            $\mathcal{P} \leftarrow \mathcal{P} \cup \{p\}$
        end if
        if $q \notin \mathcal{P}$ **then**
            $\mathcal{P} \leftarrow \mathcal{P} \cup \{q\}$
        end if
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{w = p \oplus q\}$
    end while
    $\mathcal{W} \leftarrow \mathcal{P}$
    $\mathcal{P} \leftarrow \phi$
end while
return $\mathcal{C}$

3 New Techniques for Heuristics

In this section, in order to further improve the previous forward search and backward search, we propose two heuristic techniques based on the ideas of splitting the output set and extending the base set, which are called the division optimization technique and extending base technique, respectively. The applications of these techniques will be introduced in our new framework in Section 4.

3.1 Division Optimization Technique

The division optimization technique takes the division of output set into account. For the output set $\mathcal{O} = \{y_0, y_1, \ldots, y_{m-1}\}$, we observe that the next candidate for the heuristic algorithm is usually dependent on the output set $\mathcal{O}$, which means that the results are related to specific output sets. However, previous methods treat $\mathcal{O}$ as a whole. Thus, the division of the output set may provide more possibilities.

Rational of division optimization technique. Suppose that we have the base set $\mathcal{B} = \{t_0, t_1, \ldots, t_{n-1}\}$. Our goal is to search for a circuit from $\mathcal{B}$ to $\mathcal{O}$. Usually, the search space $\mathcal{S}_\mathcal{B}$ for the next candidates is too large to traverse all the choices. The heuristic algorithm is used to reduce the search space. We use $\mathcal{H}_f$ to define Rule 1 and Rule 2 in the forward search algorithm. In order to implement $\mathcal{O}$, the search space for next choice is expressed as $\mathcal{H}_f^\mathcal{O}(\mathcal{S}_\mathcal{B})$. In addition, there exist some limitations for the values, such as the minimum depth saved in $\mathcal{L}$.

Now, we formalize the division optimization technique. The technique splits the output set into two disjoint sets and optimizes them in order. The output set $\mathcal{O}$ can be split into two different sets, where

$$\mathcal{O} = \mathcal{O}_0 \cup \mathcal{O}_1, \ \mathcal{O}_0 \cap \mathcal{O}_1 = \phi. \tag{6}$$

The initial base set $\mathcal{B}$ is $\{t_0, t_1, \ldots, t_{n-1}\}$. We first optimize $\mathcal{O}_0$ and generate the updated base set and circuit. Then, $\mathcal{O}_1$ is optimized based on the newly produced base set and circuit from $\mathcal{O}_0$. Note that the optimization order of $\mathcal{O}_0$ and $\mathcal{O}_1$ may affect results. Thus, we can traverse all possible combinations. The complete algorithm to use the division optimization technique in the forward search can be seen in Algorithm 3.

Applying division optimization technique to M_1. The output set $\mathcal{O}$ of M_1 can be split into two different output sets,

$$\mathcal{O}_0 = \{y_2, y_3, y_4\}, \mathcal{O}_1 = \{y_0, y_1\}.$$

Algorithm 3 Division Optimization Technique for the Forward Search

Input: A matrix M over $M_{m\times n}$
Output: A circuit $\mathcal{C}$ to implement M

Initial the output set $\mathcal{O} \leftarrow \{y_0, y_1, \ldots, y_{m-1}\}$
$\mathcal{O} = \mathcal{O}_0 \cup \mathcal{O}_1,\ \mathcal{O}_0 \cap \mathcal{O}_1 = \phi$
Initial the base set $\mathcal{B} \leftarrow \{t_0, t_1, \ldots, t_{n-1}\}$
Initial the circuit $\mathcal{C} \leftarrow \phi$
for $k \in [0, 1]$ **do** ▷ optimizing $\mathcal{O}_0$ and $\mathcal{O}_1$ in order
    while $\mathcal{O}_k \neq \phi$ **do**
        Randomly choose a new value $t_{|\mathcal{B}|} = t_i \oplus t_j$ from $\mathcal{H}_f^{\mathcal{O}_k}(\mathcal{S}_\mathcal{B})$
        if $t_{|\mathcal{B}|} \in \mathcal{O}_k$ **then**
            $\mathcal{O}_k \leftarrow \mathcal{O}_k / \{t_{|\mathcal{B}|}\}$
        end if
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{t_{|\mathcal{B}|} = t_i \oplus t_j\}$
        $\mathcal{B} \leftarrow \mathcal{B} \cup \{t_{|\mathcal{B}|}\}$
    end while
end for
return $\mathcal{C}$

Firstly, we apply the forward search to optimize $\mathcal{O}_0$ with the base set $\mathcal{B}_0$ is also $\{t_0, t_1, t_2, t_3, t_4, t_5\}$. The obtained circuit $\mathcal{C}_0$ to implement $\mathcal{O}_0$ is as follows,

$$\begin{aligned} t_6 &= t_0 \oplus t_1, \\ t_7 &= t_2 \oplus t_3, \\ t_9 &= t_6 \oplus t_7 // y_2, \\ t_{10} &= t_4 \oplus t_9 // y_3, \\ t_{11} &= t_5 \oplus t_9 // y_4. \end{aligned}$$

Then, we can utilize the above generated circuit $\mathcal{C}_0$ to optimize another output set $\mathcal{O}_1$ with the forward search. The current base set $\mathcal{B}_1$ is $\{t_0, t_1, t_2, t_3, t_4, t_5, \boldsymbol{t_6}, \boldsymbol{t_7}, \boldsymbol{t_9}, \boldsymbol{t_{10}}, \boldsymbol{t_{11}}\}$. The circuit $\mathcal{C}_1$ to implement $\mathcal{O}_1$ is as follows,

$$\begin{aligned} & t_6 // y_0, \\ & t_8 = t_2 \oplus t_6 // y_1. \end{aligned}$$

We merge these two circuits and generate the new circuit to implement $\mathcal{O}$ with 6 `XOR` gates (see Table 4). Compared with Table 2, instead of the distance considered, the new circuit takes both the depth and the `XOR` number into account. As a result, one `XOR` gate is reduced.

The reason that the forward search misses the better circuit and will never find it lies in Rule 2 of the BP algorithm. We perform `XOR` operations on every unique pair of values in $\mathcal{B}$. If one choice can generate one output signal, it will be chosen first. y_0, y_1, and y_2 must be generated in order. However, y_2 cannot be used to produce the new values as $\mathcal{D}(y_2)$ is 3. Therefore, the algorithm has to use t_9 and t_{11} to generate t_{10} and t_{12}, respectively. Using the division optimization technique, the depth of y_2 is only 2 and can be used in subsequent circuits.

Table 4: The new implementation of M_1.

No.	Operation	Depth	New value	New dist
1	$t_6 = t_0 \oplus t_1 // y_0$	1	$t_6 = [1,1,0,0,0,0]$	$[\mathbf{0},\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{3}] = 9$
2	$t_7 = t_2 \oplus t_3$	1	$t_7 = [0,0,1,1,0,0]$	$[0,1,\mathbf{1},\mathbf{2},\mathbf{2}] = 6$
3	$t_8 = t_2 \oplus t_6 // y_1$	2	$t_8 = [1,1,1,0,0,0]$	$[0,\mathbf{0},1,2,2] = 5$
4	$t_9 = t_6 \oplus t_7 // y_2$	2	$t_9 = [1,1,1,1,0,0]$	$[0,0,\mathbf{0},\mathbf{1},\mathbf{1}] = 2$
5	$t_{10} = t_4 \oplus t_9 // y_3$	3	$t_{10} = [1,1,1,1,1,0]$	$[0,0,0,\mathbf{0},1] = 1$
6	$t_{11} = t_5 \oplus t_9 // y_4$	3	$t_{11} = [1,1,1,1,0,1]$	$[0,0,0,0,\mathbf{0}] = 0$

3.2 Extending Base Technique

Motivation for extending base set. In most cases, $|\mathcal{H}_f(\mathcal{S}_\mathcal{B})|$ is smaller than the complete search space $|\mathcal{S}_\mathcal{B}|$, some choices c are missed ($c \in \mathcal{S}_\mathcal{B}$ and $c \notin \mathcal{H}_f(\mathcal{S}_\mathcal{B})$). In order to illustrate it, we take M_1 as an example. For the original forward search for M_1, the whole search space $\mathcal{S}_\mathcal{B}$ for next candidate and the corresponding distance are shown in Table 5.

Table 5: $\mathcal{S}_\mathcal{B}$ and the corresponding distance.

New value	New dist	New value	New dist	New value	New dist
$t_6 = t_0 \oplus t_1$	$[0,1,2,3,3] = 9$	$t_7 = t_0 \oplus t_2$	$[1,1,2,3,3] = 10$	$t_8 = t_0 \oplus t_3$	$[1,2,2,3,3] = 11$
$t_9 = t_0 \oplus t_4$	$[1,2,3,3,4] = 13$	$t_{10} = t_0 \oplus t_5$	$[1,2,3,4,3] = 13$	$t_{11} = t_1 \oplus t_2$	$[1,1,2,3,3] = 10$
$t_{12} = t_1 \oplus t_3$	$[1,2,2,3,3] = 11$	$t_{13} = t_1 \oplus t_4$	$[1,2,3,3,4] = 13$	$t_{14} = t_1 \oplus t_5$	$[1,2,3,4,3] = 13$
$t_{15} = t_2 \oplus t_3$	$[1,2,2,3,3] = 11$	$t_{16} = t_2 \oplus t_4$	$[1,2,3,3,4] = 13$	$t_{17} = t_2 \oplus t_5$	$[1,2,3,4,3] = 13$
$t_{18} = t_3 \oplus t_4$	$[1,2,3,3,4] = 13$	$t_{19} = t_3 \oplus t_5$	$[1,2,3,4,3] = 13$	$t_{20} = t_4 \oplus t_5$	$[1,2,3,4,4] = 14$

According to the rules, $t_6 = t_0 \oplus t_1$ has the smallest distance and $\mathcal{H}_f(\mathcal{S}_\mathcal{B})$ only contain one candidate:

$$\{t_6 = t_0 \oplus t_1\}.$$

The choice $c' \notin \mathcal{H}_f(\mathcal{S}_\mathcal{B})$, has been discarded. Unless an exhaustive search has proceeded, it is difficult to predict whether discarded choices can lead to a better circuit.

In this way, only t_6 will be put into the new base set and used in the next optimization. With the impact of t_6, some candidates may never be chosen by the algorithm. Thus,in order to provide more possibilities, we put forward the extending base set.

For example, we can choose the candidates whose distance is less than 12. Therefore, t_7, t_8, t_{11}, t_{12} and t_{15} are chosen as candidates. If we choose $t_{15} = t_2 \oplus t_3$ as the next base value, the base set is extended as $\{t_0, t_1, t_2, t_3, t_4, t_5, \boldsymbol{t_{15}}\}$ and new *Dist* is $[1,2,2,3,3]$. We use the forward search to generate the circuit (see Table 6). The new circuit reduces one `XOR` gate compared with the original forward search in Table 2.

Table 6: The new circuit of M_1 based on the restricted base technique.

No.	Operation	Depth	New value	New dist
1	$\boldsymbol{t_{15} = t_2 \oplus t_3}$	1	$t_{15} = [0, 0, 1, 1, 0, 0]$	$[1, 2, \mathbf{2}, \mathbf{3}, \mathbf{3}] = 11$
2	$t_6 = t_0 \oplus t_1 // y_0$	1	$t_6 = [1, 1, 0, 0, 0, 0]$	$[\mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{2}, \mathbf{2}] = 6$
3	$t_7 = t_2 \oplus t_6 // y_1$	2	$t_7 = [1, 1, 1, 0, 0, 0]$	$[0, \mathbf{0}, 1, 2, 2] = 5$
4	$t_8 = t_6 \oplus t_{15} // y_2$	2	$t_8 = [1, 1, 1, 1, 0, 0]$	$[0, 0, \mathbf{0}, \mathbf{1}, \mathbf{1}] = 2$
5	$t_9 = t_4 \oplus t_8 // y_3$	3	$t_9 = [1, 1, 1, 1, 1, 0]$	$[0, 0, 0, \mathbf{0}, 1] = 1$
6	$t_{10} = t_5 \oplus t_8 // y_4$	3	$t_{10} = [1, 1, 1, 1, 0, 1]$	$[0, 0, 0, 0, \mathbf{0}] = 0$

Rational of extending base technique. For $\mathcal{H}_f$, $|\mathcal{S}_\mathcal{B}/\mathcal{H}_f(\mathcal{S}_\mathcal{B})|$ may be too large to traverse all the candidates. Thus, we provide a solution with less search space. We define $\mathcal{H}_b$ as the rules of the backward search. In order to extend the base set of $\mathcal{H}_f$, we can generate a circuit $\mathcal{C}'$ for the target matrix M by $\mathcal{H}_b$. Every value in $\mathcal{C}'$ is contained in the additional search space $\mathcal{S}_b$. From $\mathcal{S}_b$, we choose a subset s_b and extend the base set to $\mathcal{B} \cup s_b$. Then, we optimize the output set $\mathcal{O}$ with the extended base set $\mathcal{B} \cup s_b$. The extending base technique is implemented in Algorithm 4. We can control any subset s_b to extend the base set.

Algorithm 4 Extending Base Technique for the Forward Search

Input: A matrix M over $M_{m\times n}$, different heuristic algorithms $\mathcal{H}_f$ and $\mathcal{H}_b$
Output: A circuit $\mathcal{C}$ to implement M

Initial the output set $\mathcal{O} \leftarrow \{y_0, y_1, \ldots, y_{m-1}\}$
Initial the base set $\mathcal{B} \leftarrow \{t_0, t_1, \ldots, t_{n-1}\}$
Initial the circuit $\mathcal{C} \leftarrow \phi$
Calculate $\mathcal{S}_b$ based on the backward search $\mathcal{H}_b$ ▷ the additional search space
Choose a subset $s_b \subset \mathcal{S}_b$ ▷ choosing a subset of $\mathcal{S}_b$
$\mathcal{B} \leftarrow \mathcal{B} \cup s_b$ ▷ extending the base set using s_b
while $\mathcal{O} \neq \phi$ **do**
    randomly choose a new value $t_{|\mathcal{B}|} = t_i \oplus t_j$ from $\mathcal{H}_f^\mathcal{O}(\mathcal{S}_\mathcal{B})$
    if $t_{|\mathcal{B}|} \in \mathcal{O}$ **then**
        $\mathcal{O} \leftarrow \mathcal{O}/\{t_{|\mathcal{B}|}\}$
    end if
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{t_{|\mathcal{B}|} = t_i \oplus t_j\}$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{t_{|\mathcal{B}|}\}$
end while
return $\mathcal{C}$

3.3 Applying to the Backward Search

We have introduced our new techniques based on the forward search $\mathcal{H}_f$. Actually, the techniques can also be used to improve the backward search $\mathcal{H}_b$, where we replace the output set $\mathcal{O}$ and the base set $\mathcal{B}$ with the working set $\mathcal{W}$ and the

predecessor set $\mathcal{P}$, respectively. We just adjust the techniques for the backward search.

For the division optimization technique, we can also optimize $\mathcal{O}_0$ and $\mathcal{O}_1$ in order. For the extending base technique, we combine the technique with the backward search in Algorithm 5. We take M_2 as an example to illustrate it.

Algorithm 5 Extending Base Technique for the Backward Search

Input: A matrix M over $M_{m\times n}$, different heuristic algorithms $\mathcal{H}_f$ and $\mathcal{H}_b$
Output: A circuit $\mathcal{C}$ to implement M

$\mathcal{W} \leftarrow \{y_0, y_1, \ldots, y_{m-1}\}$ ▷ the working set
$\mathcal{X} \leftarrow \{t_0, t_1, \ldots, t_{n-1}\}$ ▷ the input set
$\mathcal{P} \leftarrow \phi$ ▷ the predecessor set
$\mathcal{C} \leftarrow \phi$ ▷ the circuit
Calculate $\mathcal{S}_f$ based on the forward search $\mathcal{H}_f$ ▷ the additional search space
Choose a subset $s_f \subset \mathcal{S}_f$ ▷ choosing a subset of $\mathcal{S}_f$
$\mathcal{P} \leftarrow \mathcal{P} \cup s_f$ ▷ extending the base set using s_f
while $\mathcal{X} \cup \mathcal{W} \neq \mathcal{X}$ **do**
    while $\mathcal{W} \neq \phi$ **do**
        Choose a value $w \in \mathcal{W}$ by Rule 1-Rule 5 and split w by $w = p \oplus q$.
        if $p \notin \mathcal{P}$ **then**
            $\mathcal{P} \leftarrow \mathcal{P} \cup \{p\}$
        end if
        if $q \notin \mathcal{P}$ **then**
            $\mathcal{P} \leftarrow \mathcal{P} \cup \{q\}$
        end if
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{w = p \oplus q\}$
    end while
    $\mathcal{W} \leftarrow \mathcal{P}$
    $\mathcal{P} \leftarrow \phi$
end while
return $\mathcal{C}$

After the initialization, we have the predecessor set $\mathcal{P} = \phi$, the working set $\mathcal{W} = \{y_0, y_1, y_2, y_3\}$, and $\mathcal{X} = \{t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$.

We find the candidate

$$y_1 = t_4 \oplus y_2$$

will never be chosen by $\mathcal{H}_b$. While the candidate $y_1 = t_4 \oplus y_2$ belongs to the search space of $\mathcal{H}_f$. This means $\{t_4, y_2\} \in \mathcal{S}_f$. We just choose $s_f = \{t_4, y_2\} \in \mathcal{S}_f$ to update $\mathcal{P}$. We can add $y_1 = t_4 \oplus y_2$ into the circuit and let $\mathcal{P} = \mathcal{P} \cup \{t_4, y_2\}$. The working set is $\mathcal{W} = \{y_0, y_3\}$, while the new predecessor set is $\mathcal{P} = \{y_2, t_4\}$. The running process is shown in Table 7.

In Table 3, t_{10} is used to split y_2. However, we do not generate t_{10}. We can split y_2 by $y_2 = t_7 \oplus y_3$. t_8 is also split by $t_8 = t_{11} \oplus t_{12}$. The new circuit is shown in Table 7, which saves one `XOR` gate. We find that the additional restrictions relax the depth limit of y_1. $\mathcal{D}(y_1) = 3$, but $\mathcal{D}_{min}(y_1) = 2$. According to previous rules, $\mathcal{D}(y_1)$ must be 2.

Table 7: The new implementation of M_2.

No.	Operation	Depth	New value	Minimum depth
1	$\boldsymbol{y_1 = t_4 \oplus y_2}$	**3**	$y_1 = [0,0,0,0,1,1,1,1]$	$\mathcal{D}_{min}(y_1) = 2$
2	$y_0 = t_8 \oplus t_9$	3	$y_0 = [1,1,1,1,1,1,0,0]$	$\mathcal{D}_{min}(y_0) = 3$
3	$\boldsymbol{y_2 = t_7 \oplus y_3}$	2	$y_2 = [0,0,0,0,0,1,1,1]$	$\mathcal{D}_{min}(y_2) = 2$
4	$t_8 = t_{11} \oplus t_{12}$	2	$t_8 = [1,1,1,1,0,0,0,0]$	$\mathcal{D}_{min}(t_8) = 2$
5	$t_{11} = t_0 \oplus t_1$	1	$t_{11} = [1,1,0,0,0,0,0,0]$	$\mathcal{D}_{min}(t_{11}) = 1$
6	$t_{12} = t_2 \oplus t_3$	1	$t_{12} = [0,0,1,1,0,0,0,0]$	$\mathcal{D}_{min}(t_{12}) = 1$
7	$t_9 = t_4 \oplus t_5$	1	$t_9 = [0,0,0,0,1,1,0,0]$	$\mathcal{D}_{min}(t_9) = 1$
8	$y_3 = t_5 \oplus t_6$	1	$y_3 = [0,0,0,0,0,1,1,0]$	$\mathcal{D}_{min}(y_3) = 1$

4 General Framework of Optimization

We have introduced the division optimization technique and the extending base technique. These techniques can be utilized to improve heuristic algorithms. For a given circuit from the forward search (or backward search), the division optimization technique can also be used to divide the partial circuit besides the output signals. In Table 2, we can choose

$$t_{10} = t_7 \oplus t_9,\ t_{11} = t_3 \oplus t_5,\ t_{12} = t_7 \oplus t_{11}$$

as the new target set, where t_{11} is not the output signal. In addition, we can extend the base set by appending additional base into the base set. In this section, we will provide a framework to use the backward search (or forward search) to optimize the middle part based on the extending base set to update the given circuit.

4.1 Division of a Given Circuit $\mathcal{C}$

The circuit is split into three parts. The first part is the base part $\mathcal{B}'_{\mathcal{C}}$, which consists of the previous base set and additional values in $\mathcal{C}$. The second part is the target set $\mathcal{O}'_{\mathcal{C}}$, which contains some intermediate values and output values. The rest of the circuit is the unrelated part $\mathcal{U}'_{\mathcal{C}}$. Then, we can generate a new circuit to optimize $\mathcal{O}'_{\mathcal{C}}$ based on the base part $\mathcal{B}'_{\mathcal{C}}$.

In order to split the circuit, we first introduce the definition of topological ordering. The problem of finding a topological ordering can be solved in linear time by Kahn's algorithm [18].

Definition 2. *Given a circuit $\mathcal{C}$, the topological ordering of a circuit C is an ordering of its values into a sequence, which is denoted as $\mathcal{T}_{\mathcal{C}}$. For every* `XOR` *gate $t_a = t_b \oplus t_c$, the input values t_b and t_c of the gate occur earlier in the sequence than the output value t_a.*

M_1 is taken as an example. We use $X_{a,b,c}$ to represent the `XOR` gate $t_a = t_b \oplus t_c$, use $X_{\underline{a},b,c}$ to denote that t_a is the output value. The circuit $\mathcal{C}$ of M_1 in Table 2

is:

$$X_{\underline{6},0,1}, X_{\underline{7},6,2}, X_{\underline{8},7,3}, X_{9,3,4}, X_{\underline{10},7,9}, X_{11,3,5}, X_{\underline{12},7,11}.$$

The input set $\mathcal{B}$ is $\{t_0, t_1, t_2, t_3, t_4, t_5\}$, the output set $\mathcal{O}$ is $\{t_6, t_7, t_8, t_{10}, t_{12}\}$. The topological ordering is

$$\mathcal{T}_{\mathcal{C}} = t_0, t_1, t_2, t_3, t_4, t_5, t_{\underline{6}}, t_9, t_{11}, t_{\underline{7}}, t_{\underline{8}}, t_{\underline{10}}, t_{\underline{12}}, \tag{7}$$

where $t_{\underline{a}}$ represents that t_a is the output value.

Based on the topological ordering $\mathcal{T}_{\mathcal{C}}$, we split $\mathcal{C}$ into three parts (see Figure 1).

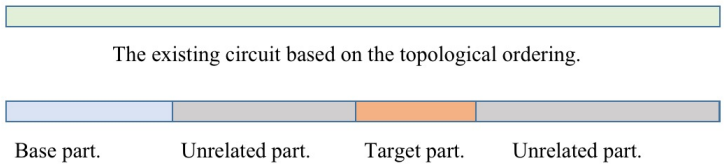

Fig. 1: Division of the given circuit.

- **Base part $\mathcal{B}'_{\mathcal{C}}$.** The base set $\mathcal{B}$ is $\{t_0, \ldots, t_{n-1}\}$. We choose a subset $\mathcal{T}_{sub} \subset \mathcal{T}_{\mathcal{C}}/\mathcal{B}$ to extend the base set $\mathcal{B}$ to produce the base part $\mathcal{B}'_{\mathcal{C}}$. We have the base part $\mathcal{B}'_{\mathcal{C}} = \mathcal{B} \cup \mathcal{T}_{sub}$.
- **Target part $\mathcal{O}'_{\mathcal{C}}$.** We choose a subset from $\mathcal{T}_{\mathcal{C}}/\mathcal{B}'_{\mathcal{C}}$ as the optimized target part $\mathcal{O}'_{\mathcal{C}}$.
- **Unrelated part $\mathcal{U}'_{\mathcal{C}}$.** The unrelated part $\mathcal{U}'_{\mathcal{C}}$ is $\mathcal{T}_{\mathcal{C}}/\{\mathcal{B}'_{\mathcal{C}} \cup \mathcal{O}'_{\mathcal{C}}\}$.

In the example of M_1, we choose $\mathcal{T}_{sub} = \{t_6\}$ and the base part $\mathcal{B}'_{\mathcal{C}}$ is $\{t_0, t_1, t_2, t_3, t_4, t_5, t_6\}$. Then, we choose the target part $\mathcal{O}'_{\mathcal{C}}$ is $\{t_8, t_{10}, t_{12}\}$. The unrelated part $\mathcal{U}'_{\mathcal{C}}$ is $\{t_7, t_9, t_{11}\}$.

4.2 Updating the Circuit $\mathcal{C}$ by Optimizing $\mathcal{O}'_{\mathcal{C}}$ with $\mathcal{B}'_{\mathcal{C}}$

Given a division of the circuit $\mathcal{C}$, we optimize the target part $\mathcal{O}'_{\mathcal{C}}$ with the base part $\mathcal{B}'_{\mathcal{C}}$ to generate a new circuit. Because the new circuit only contains partial information of the given circuit $\mathcal{C}$, we call it $\mathcal{C}_{part}$. In order to update $\mathcal{C}$, the following three steps will proceed.

- **Step 1.** Optimize $\mathcal{O}'_{\mathcal{C}}$ with $\mathcal{B}'_{\mathcal{C}}$ to generate the partial circuit $\mathcal{C}_{part}$.
- **Step 2.** Merge the partial circuit $\mathcal{C}_{part}$ and original circuit $\mathcal{C}$ into the new circuit $\mathcal{C}'$.
- **Step 3.** Remove redundant `XOR` gates from $\mathcal{C}'$ and set $\mathcal{C} = \mathcal{C}'$.

Generate the partial circuit $\mathcal{C}_{part}$. There exist two modes to optimize $\mathcal{O}'_{\mathcal{C}}$ with $\mathcal{B}'_{\mathcal{C}}$ to generate the partial circuit $\mathcal{C}_{part}$. Suppose that the given circuit $\mathcal{C}$ is generated by the forward search $\mathcal{H}_f$, we have two modes to build $\mathcal{C}_{part}$. Mode 1 is to use the same heuristic algorithm $\mathcal{H}_f$ to generate $\mathcal{C}_{part}$. Mode 2 is to use another heuristic algorithm $\mathcal{H}_b$ to generate $\mathcal{C}_{part}$.

For M_1, the circuit $\mathcal{C}$ is generated by $\mathcal{H}_f$. Let the base part $\mathcal{B}'_{\mathcal{C}}$ and the target part $\mathcal{O}'_{\mathcal{C}}$ be $\{t_0, t_1, t_2, t_3, t_4, t_5, t_6\}$ and $\{t_8, t_{10}, t_{12}\}$, respectively. Using the forward search, we generate the partial circuit $\mathcal{C}_{part}$,

$$X_{13,2,3},\ X_{\underline{8},6,13},\ X_{\underline{10},8,4},\ X_{\underline{12},8,5}.$$

Merging $\mathcal{C}_{part}$ and $\mathcal{C}$ into the new circuit $\mathcal{C}'$. After getting $\mathcal{C}_{part}$, initialize $\mathcal{C}' = \mathcal{C}$. We need to merge $\mathcal{C}_{part}$ and $\mathcal{C}$ into the new circuit $\mathcal{C}'$. For each value $t_a \in \mathcal{T}_{\mathcal{C}_{part}}$, there exist two cases, $t_a \notin \mathcal{T}_{\mathcal{C}}$ or $t_a \in \mathcal{T}_{\mathcal{C}}$. For the first case, we put the corresponding `XOR` gates into the new circuit $\mathcal{C}'$. For the second case, we use the `XOR` gate in $\mathcal{C}_{part}$ to replace the previous `XOR` gate in $\mathcal{C}'$. The processes to merge $\mathcal{C}_{part}$ and $\mathcal{C}$ into the new circuit $\mathcal{C}'$ can be seen in Algorithm 6.

We still take M_1 as an example. The topological ordering of $\mathcal{T}_{\mathcal{C}_{part}}$ is

$$\mathcal{T}_{\mathcal{C}_{part}} = t_0, t_1, t_2, t_3, t_4, t_5, t_{\underline{6}}, \boldsymbol{t_{13}}, t_{\underline{8}}, t_{\underline{10}}, t_{\underline{12}},$$

where $t_{13} \notin \mathcal{T}_{\mathcal{C}}$ (see Equation (7)). The corresponding `XOR` gate is $X_{13,2,3}$. Thus, we produce the new circuit $\mathcal{C}'$ is

$$\mathcal{C}' = X_{\underline{6},0,1}, \boldsymbol{X_{13,2,3}}, X_{\underline{7},6,2}, X_{\underline{8},7,3}, X_{9,3,4}, X_{\underline{10},7,9}, X_{11,3,5}, X_{\underline{12},7,11}. \quad (8)$$

We notice that $t_8, t_{10}, t_{12} \in \mathcal{T}_{\mathcal{C}_{part}}$. So we replace t_8, t_{10}, and t_{12} in $\mathcal{T}_{\mathcal{C}}$ with the corresponding ones in $\mathcal{T}_{\mathcal{C}_{part}}$.

$$\mathcal{C}' = X_{\underline{6},0,1}, X_{13,2,3}, X_{\underline{7},6,2}, \boldsymbol{X_{\underline{8},6,13}}, X_{9,3,4}, \boldsymbol{X_{\underline{10},8,4}}, X_{11,3,5}, \boldsymbol{X_{\underline{12},8,5}}. \quad (9)$$

Removing redundant `XOR` gates. After finishing the merging process, the achieved circuit $\mathcal{C}'$ may have redundant `XOR` gates. For an `XOR` gate $X_{a,b,c}$, if t_a is not the output signal and t_a is not used to generate any new values, we say that t_a and $X_{a,b,c}$ are redundant. For example, in Equation (8), t_9 is used to generate t_{10}, while in Equation (9), t_9 is not used and t_{10} is generated by $t_4 \oplus t_8$, so t_9 is redundant.

We use the graph extending technique [26] to remove redundant values from given circuits. The technique use $od(t_a)$ to count the number times that t_a is used. In Equation (9), we have

$$od(t_0) = 1, od(t_1) = 1, od(t_2) = 2, od(t_3) = 3, od(t_4) = 2, od(t_5) = 2, od(t_6) = 2,$$
$$od(t_7) = 0, od(t_8) = 2, \boldsymbol{od(t_9) = 0}, od(t_{10}) = 0, \boldsymbol{od(t_{11}) = 0}, od(t_{12}) = 0, od(t_{13}) = 1.$$

We notice $od(t_9) = 0$ and $od(t_{11}) = 0$, and t_9, t_{11} are not the output values. Thus, t_9 and t_{11} are redundant. Thus, it is no need to generate these two values, so $X_{9,3,4}$ and $X_{11,3,5}$ can be removed from $\mathcal{C}'$. New circuit is

$$\mathcal{C}' = X_{\underline{6},0,1}, X_{\underline{7},6,2}, X_{13,2,3}, \boldsymbol{X_{\underline{8},6,13}}, \boldsymbol{X_{\underline{10},8,4}}, \boldsymbol{X_{\underline{12},8,5}}. \quad (10)$$

Algorithm 6 MergeCircuit()

Input: The previous circuit $\mathcal{C}$ and the partial circuit $\mathcal{C}_{part}$
Output: The new circuit $\mathcal{C}'$

Initial the additional circuit $\mathcal{C}_{add} = \phi$
Calculate the topological ordering $\mathcal{T}_{\mathcal{C}}$ and $\mathcal{T}_{\mathcal{C}_{part}}$
for each value $t_a \in \mathcal{T}_{\mathcal{C}_{part}}$ **do**
    if $t_a \notin \mathcal{T}_{\mathcal{C}}$ **then**
        Choose the corresponding `XOR` gate $X_{a,b,c} \in \mathcal{C}_{part}$
        $\mathcal{C}_{add} \leftarrow \mathcal{C}_{add} \cup \{X_{a,b,c}\}$ ▷ adding additional `XOR` gates
    end if
end for
$\mathcal{C}' = \mathcal{C} \cup \mathcal{C}_{add}$
for each value $t_a \in \mathcal{T}_{\mathcal{C}}$ **do**
    if $t_a \in \mathcal{T}_{\mathcal{C}_{part}}$ **then**
        Find the corresponding `XOR` gates $X_{a,b,c} \in \mathcal{C}$ and $X_{a,b',c'} \in \mathcal{C}_{part}$
        Use $X_{a,b',c'}$ to replace $X_{a,b,c}$ in $\mathcal{C}'$.
    end if
end for
return $\mathcal{C}'$

Then, we use $\mathcal{C}'$ to update $\mathcal{C}$. It reduces one `XOR` gate. We use the function Remove() to represent the graph extending technique in Section 4.3.

4.3 Continuous Division Strategy

Considering all the combinations of different $\mathcal{B}'_{\mathcal{C}}$ and $\mathcal{O}'_{\mathcal{C}}$, we deduce that it is infeasible to exhaust them and then try to divide. The number of all the possible combinations of $\mathcal{B}'_{\mathcal{C}}$ is $2^{|\mathcal{T}_{\mathcal{C}}|-|\mathcal{B}|}-1$, and for the fixed $\mathcal{B}'_{\mathcal{C}}$, all the possible forms of $\mathcal{O}'_{\mathcal{C}}$ is $2^{|\mathcal{T}_{\mathcal{C}}|-|\mathcal{B}'_{\mathcal{C}}|}-1$. The detailed proof is shown in Supplementary Materials B. As the circuit size is too large, we cannot traverse all the combinations. For the circuit of `AES` MixColumns with 103 `XOR` gates, the number of the combinations of $\mathcal{B}'_{\mathcal{C}}$ is $2^{71}-1$. Thus, we give the continuous division strategy, which is executed in a reasonable time.

For a given circuit $\mathcal{C}$, let $|\mathcal{B}'_{\mathcal{C}}|$ be $|\mathcal{B}|$ and $|\mathcal{O}'_{\mathcal{C}}|$ be 1. Then, we gradually increase $|\mathcal{B}'_{\mathcal{C}}|$ from $|\mathcal{B}|$ to $|\mathcal{T}_{\mathcal{C}}|$. For every fixed $\mathcal{B}'_{\mathcal{C}}$, we gradually increase $|\mathcal{O}'_{\mathcal{C}}|$ from 1 to $|\mathcal{T}_{\mathcal{C}}| - |\mathcal{B}'_{\mathcal{C}}|$. In order to reduce the search space, we always choose the consequent values in $\mathcal{T}_{\mathcal{C}}$ to extend $\mathcal{B}$ and the consequent $\mathcal{O}'_{\mathcal{C}}$. The reduction process for a given circuit is shown in Algorithm 7.

Combination with heuristics. We can combine our algorithm with any heuristics. Given the target matrix, we can generate a circuit $\mathcal{C}$ based on the forward search or the backward search. Then, we divide the circuit to generate the $\mathcal{C}_{part}$, merge $\mathcal{C}_{part}$ and $\mathcal{C}$ into the new circuit $\mathcal{C}'$, and update $\mathcal{C}$ after removing the redundant `XOR` gates. A tradeoff between the running times of heuristics and the number of combinations of different $\mathcal{B}'_C$ and $\mathcal{O}'_C$ need to be considered.

Algorithm 7 Continuous Division Strategy

Input: A circuit $\mathcal{C}$ to implement the target matrix M
Output: A new circuit $\mathcal{C}_{best}$ to implement M

The best circuit $\mathcal{C}_{best} \leftarrow \mathcal{C}$
Put the input values and the output values into $\mathcal{B}$ and $\mathcal{O}$, respectively
Calculate $\mathcal{T}_C = \mathcal{T}_C[1]\mathcal{T}_C[2]\dots\mathcal{T}_C[|\mathcal{T}_C|]$
for pre from $|\mathcal{B}|$ to $|\mathcal{T}_C| - 1$ **do**
    $\mathcal{B}'_C \leftarrow \mathcal{T}_C[1, pre]$
    for tar from 1 to $|\mathcal{T}_C| - pre$ **do**
        for $site$ from $pre + 1$ to $|\mathcal{T}_C| - tar + 1$ **do**
            $\mathcal{O}'_C \leftarrow \mathcal{T}_C[site, site + tar]$
            $\mathcal{C}_{part} \leftarrow \mathcal{H}_f^{\mathcal{O}'_C}(\mathcal{B}'_C)$ or $\mathcal{H}_b^{\mathcal{O}'_C}(\mathcal{B}'_C)$
            $\mathcal{C}' = \text{MergeCircuit}(\mathcal{C}, \mathcal{C}_{part})$ ▷ Algorithm 6
            $\mathcal{C}' = \text{Remove}(\mathcal{C}')$ ▷ using the graph extending graph
            if $|\mathcal{C}'| < |\mathcal{C}_{best}|$ **then**
                $\mathcal{C}' \leftarrow \mathcal{C}_{best}$
            end if
        end for
    end for
end for
return $\mathcal{C}_{best}$

5 Results and Comparisons

In this section, we provide different experiments for our framework. The source codes are available at https://github.com/QunLiu-sdu/Improved-Heuristics-for-Low-latency.

5.1 The AES MixColumns

We first apply our framework to the matrix used in AES MixColumns. In [21], a circuit of AES MixColumns is reported with 105 XORs and depth 3 (105/3). Subsequently, in [6,25], the result is decreased to 103/3. However, even after running their algorithms more time, no better results can be found.

We run Algorithm 7 to optimize the circuit given in [25]. The algorithm has proceeded for five days of CPU time, which is the same as [30] in CHES 2020. Finally, we achieved the implementation with 102 XOR gates and depth 3, which is the best result until now. We provide a new implementation of AES MixColumns with 102 XORs and depth 3 in Table 8. Recent results are listed in Table 9.

5.2 Hardware Implementation

Our algorithm aims at finding optimized implementation in circuit size, power consumption, and latency. These criteria are closely related to the standard cell library. In this respect, we synthesize the implementations of AES MixColumns

Table 8: An implementation of AES MixColumns with 102 XOR operations. Here $t_0, t_1, t_2, \ldots, t_{31}$ are input values and $y_0, y_1, y_2, \ldots, y_{31}$ are the 32 output values.

No.	Operation	Depth	No.	Operation	Depth	No.	Operation	Depth
1	$t_{140} = t_{18} + t_9$	1	35	$t_{64} = t_4 + t_{20}$	1	69	$t_{98} = t_{84} + t_{96}//y_8$	3
2	$t_{32} = t_5 + t_{13}$	1	36	$t_{65} = t_{64} + t_{53}$	2	70	$t_{99} = t_9 + t_{25}$	1
3	$t_{33} = t_{21} + t_{29}$	1	37	$t_{66} = t_{12} + t_{20}$	1	71	$t_{100} = t_{95} + t_{99}$	2
4	$t_{34} = t_{15} + t_{30}$	1	38	$t_{67} = t_5 + t_{66}$	2	72	$t_{101} = t_{100} + t_{46}//y_1$	3
5	$t_{35} = t_7 + t_{16}$	1	39	$t_{68} = t_{33} + t_{67}//y_{13}$	3	73	$t_{102} = t_1 + t_{17}$	1
6	$t_{36} = t_{23} + t_{24}$	1	40	$t_{69} = t_{67} + t_{56}//y_{21}$	3	74	$t_{138} = t_{10} + t_{102}$	2
7	$t_{37} = t_1 + t_{18}$	1	41	$t_{70} = t_{14} + t_{21}$	1	75	$t_{103} = t_{95} + t_{102}$	2
8	$t_{38} = t_{17} + t_{26}$	1	42	$t_{71} = t_5 + t_{70}$	2	76	$t_{104} = t_{103} + t_{79}//y_9$	3
9	$t_{137} = t_{38} + t_{140}$	2	43	$t_{72} = t_{71} + t_{40}//y_{30}$	3	77	$t_{105} = t_4 + t_{28}$	1
10	$t_{39} = t_6 + t_{22}$	1	44	$t_{73} = t_{18} + t_{23}$	1	78	$t_{106} = t_{105} + t_{21}$	2
11	$t_{40} = t_{39} + t_{33}$	2	45	$t_{74} = t_{11} + t_{27}$	1	79	$t_{107} = t_{56} + t_{106}//y_5$	3
12	$t_{41} = t_{14} + t_{31}$	1	46	$t_{75} = t_{73} + t_{74}$	2	80	$t_{108} = t_{32} + t_{106}//y_{29}$	3
13	$t_{42} = t_{41} + t_{39}$	2	47	$t_{76} = t_3 + t_{19}$	1	81	$t_{109} = t_{19} + t_{23}$	1
14	$t_{43} = t_7 + t_{15}$	1	48	$t_{77} = t_{73} + t_{76}$	2	82	$t_{110} = t_{105} + t_{109}$	2
15	$t_{44} = t_{43} + t_{41}$	2	49	$t_{78} = t_{16} + t_{23}$	1	83	$t_{111} = t_{110} + t_{93}//y_{20}$	3
16	$t_{45} = t_0 + t_{17}$	1	50	$t_{79} = t_{78} + t_{25}$	2	84	$t_{114} = t_{137} + t_{138}//y_2$	3
17	$t_{46} = t_7 + t_{45}$	2	51	$t_{80} = t_0 + t_8$	1	85	$t_{117} = t_2 + t_{137}//y_{10}$	3
18	$t_{47} = t_6 + t_{23}$	1	52	$t_{81} = t_{31} + t_{80}$	2	86	$t_{118} = t_{10} + t_{27}$	1
19	$t_{48} = t_7 + t_{47}$	2	53	$t_{82} = t_{81} + t_{36}//y_{16}$	3	87	$t_{119} = t_{15} + t_{118}$	2
20	$t_{49} = t_{48} + t_{44}//y_7$	3	54	$t_{83} = t_{81} + t_{35}//y_{24}$	3	88	$t_{120} = t_{77} + t_{119}//y_{11}$	3
21	$t_{50} = t_{42} + t_{48}//y_{15}$	3	55	$t_{84} = t_{78} + t_{80}$	2	89	$t_{121} = t_{11} + t_{20}$	1
22	$t_{51} = t_{34} + t_{48}//y_{31}$	3	56	$t_{85} = t_2 + t_{10}$	1	90	$t_{122} = t_{15} + t_{121}$	2
23	$t_{52} = t_{12} + t_{28}$	1	57	$t_{86} = t_{85} + t_{25}$	2	91	$t_{123} = t_{54} + t_{122}//y_4$	3
24	$t_{53} = t_3 + t_7$	1	58	$t_{87} = t_{86} + t_{38}//y_{18}$	3	92	$t_{124} = t_{110} + t_{122}//y_{12}$	3
25	$t_{54} = t_{52} + t_{53}$	2	59	$t_{88} = t_{86} + t_{37}//y_{26}$	3	93	$t_{125} = t_{11} + t_{19}$	1
26	$t_{55} = t_{13} + t_{29}$	1	60	$t_{89} = t_3 + t_{26}$	1	94	$t_{126} = t_2 + t_7$	1
27	$t_{56} = t_{52} + t_{55}$	2	61	$t_{90} = t_{89} + t_{31}$	2	95	$t_{127} = t_{125} + t_{126}$	2
28	$t_{57} = t_{30} + t_{55}$	2	62	$t_{91} = t_{75} + t_{90}//y_{19}$	3	96	$t_{128} = t_{127} + t_{119}//y_3$	3
29	$t_{58} = t_{57} + t_{40}//y_{14}$	3	63	$t_{92} = t_{12} + t_{27}$	1	97	$t_{129} = t_{127} + t_{90}//y_{27}$	3
30	$t_{59} = t_{14} + t_{22}$	1	64	$t_{93} = t_{92} + t_{31}$	2	98	$t_{130} = t_9 + t_{31}$	1
31	$t_{60} = t_{59} + t_{30}$	2	65	$t_{94} = t_{65} + t_{93}//y_{28}$	3	99	$t_{131} = t_1 + t_{24}$	1
32	$t_{61} = t_{32} + t_{60}//y_6$	3	66	$t_{95} = t_8 + t_{15}$	1	100	$t_{132} = t_{130} + t_{131}$	2
33	$t_{62} = t_{40} + t_{60}//y_{22}$	3	67	$t_{96} = t_{24} + t_{95}$	2	101	$t_{133} = t_{132} + t_{79}//y_{17}$	3
34	$t_{63} = t_{44} + t_{60}//y_{23}$	3	68	$t_{97} = t_{96} + t_{35}//y_0$	3	102	$t_{134} = t_{132} + t_{46}//y_{25}$	3

Table 9: The circuits of matrix used in AES MixColumns. The cost is XOR/depth. Here 102/3 means it requires 102 XOR gates with depth 3.

Source	[19]	[30]	[31]	[23]
XORs/Depth	97/8	94/6	92/6	91/7
Source	[21]	[25]	[6]	**This paper**
XORs/Depth	**105/3**[a]	**103/3**[a]	**103/3**[a]	**102/3**[a]

[a] With the limitation of minimum depth.

with UMC 55 nm library and show their performance in hardware (see Table 10). The logic synthesis is performed with Synopsys Design Compiler version R-2020.09-SP4 (using the compile_ultra and compile_ultra -no_autoungroup commands), and simulation is done in Mentor Graphics ModelSim SE v10.2c. Our AES MixColumns implementation has more advantages than other low-latency circuits.

Table 10: The results of AES MixColumns in UMC 55 nm library.

Type	**Latency** (us)	**Area** (GE)	**Power** (uW)
[23][a]	0.52	227.5	17.5
[26][b]	0.65	220.0	16.0
[25][c]	0.28	257.5	15.9
This paper[d]	**0.27**	**255.0**	**15.6**

[a] Using 91 XORs with depth 7.
[b] Using 61 XORs and 15 3-input XOR gates.
[c] Using 103 XORs with depth 3.
[d] Using 102 XORs with depth 3.

5.3 XOR Gates of Many Proposed Matrices

In this section, we apply our algorithm to several linear layers from the literature including matrices used in many ciphers [1,2,3,5,8,10,13,14,16] and matrices independently proposed in many previous works [9,17,19,22,24,28,29].

Comparison. The comparison with [6,21,25] are listed in Table 1. For each matrix, we take no more than five days of CPU time to run Algorithm 7. Apart from AES MixColumns, eight better circuits are found by our algorithm. We bold the optimized results in the table. All the results are required to implement under the low-latency criterion.

5.4 Matrices from [21]

We apply our algorithm on 4254 matrices given in [21], which have provided the corresponding circuits with the minimum depth 3 in [21]. The Hamming weight for them is between 148-172 and the size is 32×32.

Overall improvements. As a result, we have improved about 3300 (77.57%) matrices in terms of the number of XOR gates (see Supplementary Materials C for all the results). For each Hamming weight, we can optimize the minimum XOR gates in most cases. The minimum number of XORs is decreased from 88 [21] to 85 (cf. Figure 2 and 3).

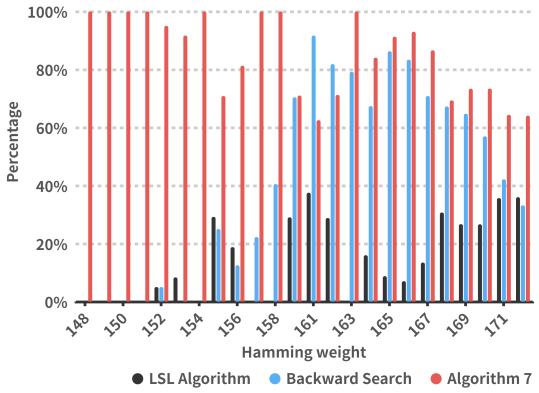

Fig. 2: Comparison of the optimized percentage with different Hamming weight.

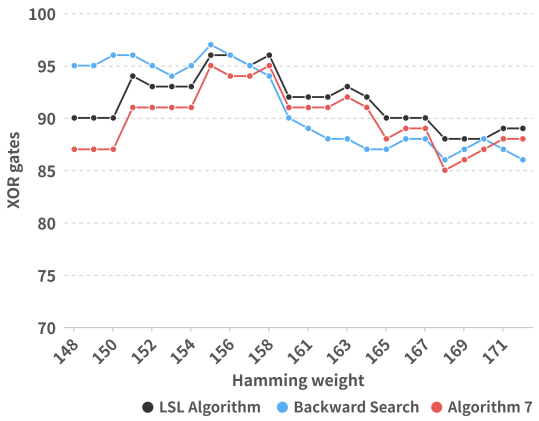

Fig. 3: Comparison of the minimum `XOR` gates with different Hamming weight.

New results for the MDS matrices. For the involutory MDS matrices with size 4×4, in which each element is in the field $\mathrm{GL}(8,\mathbb{F}_2)$, in [19], the smallest number of XORs of is 96/3. The number is decreased to 88/3 through lots of searches and new heuristics [21], which later has been improved to 86/3 in [25]. With the help of our algorithm, a new record is reported. We find a circuit requiring 85 XORs with depth 3 (see Table 11 for the comparison). The characteristic polynomial is $(x^4+x+1)^2 = x^8+x^2+1$ and the companion matrix A is

$$\begin{bmatrix} 0&0&0&0&0&0&0&1\\ 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&1\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0 \end{bmatrix}.$$

The lightest one that we find is M_3,

$$\begin{bmatrix} I_8 & I_8 & A^{-2} & A^{-2}\\ A^{10} & I_8 & A^2 & A^4\\ A^6 & I_8 & I_8 & A^6\\ A^4 & I_8 & A^4 & I_8 \end{bmatrix},$$

whose circuit is shown in Table 12.

Table 11: Comparison of the 4×4 lightest MDS matrices. The general linear group $\mathrm{GL}(n,\mathbb{F}_2)$ is formed by all invertible $n\times n$ matrices over $\mathbb{F}_2$. $A\in\mathrm{GL}(n,\mathbb{F}_2)$ is Involutory if and only if $A=A^{-1}$.

Entries	Involutory	Best depth	XORs	Source
$\mathrm{GL}(4,\mathbb{F}_2)$	✗	✗	35/6	[15]
$\mathrm{GL}(4,\mathbb{F}_2)$	✗	✓	40/3	[25]
$\mathrm{GL}(4,\mathbb{F}_2)$	✓	✗	35/8	[32]
$\mathrm{GL}(4,\mathbb{F}_2)$	✓	✓	43/3	[6], **This paper**
$\mathrm{GL}(8,\mathbb{F}_2)$	✗	✗	67/6	[15]
$\mathrm{GL}(8,\mathbb{F}_2)$	✗	✓	77/3	[15]
$\mathrm{GL}(8,\mathbb{F}_2)$	✓	✗	70/9	[32]
$\mathrm{GL}(8,\mathbb{F}_2)$	✓	✓	88/3	[21]
$\mathrm{GL}(8,\mathbb{F}_2)$	✓	✓	86/3	[25]
$\mathrm{GL}(8,\mathbb{F}_2)$	✓	✓	**85/3**	**This paper**

6 Conclusion

In this paper, we propose two new techniques, the division optimization technique and the extending base technique. We show the effect of these new tech-

Table 12: An implementation of M_3 with 85 XOR gates.

No.	Operation	Depth	No.	Operation	Depth	No.	Operation	Depth
1	$t_{32} = t_4 + t_{20}$	1	30	$t_{61} = t_{44} + t_{60}//y_4$	2	59	$t_{94} = t_{14} + t_{30}$	1
2	$t_{33} = t_5 + t_{21}$	1	31	$t_{64} = t_6 + t_{16}$	1	60	$t_{95} = t_{52} + t_{94}//y_{30}$	2
3	$t_{34} = t_6 + t_{22}$	1	32	$t_{65} = t_{50} + t_{64}//y_6$	2	61	$t_{96} = t_{15} + t_{27}$	1
4	$t_{35} = t_7 + t_{23}$	1	33	$t_{66} = t_{22} + t_{30}$	1	62	$t_{97} = t_{58} + t_{96}//y_{15}$	3
5	$t_{36} = t_2 + t_{26}$	1	34	$t_{67} = t_{40} + t_{66}$	2	63	$t_{98} = t_{15} + t_{31}$	1
6	$t_{37} = t_3 + t_{27}$	1	35	$t_{68} = t_{39} + t_{67}//y_{10}$	3	64	$t_{99} = t_{54} + t_{98}//y_{31}$	2
7	$t_{38} = t_4 + t_{28}$	1	36	$t_{69} = t_{65} + t_{67}//y_{22}$	3	65	$t_{100} = t_{18} + t_{36}$	2
8	$t_{39} = t_{10} + t_{38}$	2	37	$t_{70} = t_7 + t_{17}$	1	66	$t_{101} = t_{39} + t_{100}//y_{18}$	3
9	$t_{40} = t_0 + t_{16}$	1	38	$t_{71} = t_{51} + t_{70}//y_7$	2	67	$t_{102} = t_{19} + t_{37}$	2
10	$t_{41} = t_5 + t_{29}$	1	39	$t_{72} = t_{23} + t_{31}$	1	68	$t_{103} = t_{42} + t_{102}//y_{19}$	3
11	$t_{42} = t_{11} + t_{41}$	2	40	$t_{73} = t_{43} + t_{72}$	2	69	$t_{108} = t_6 + t_{28}$	1
12	$t_{43} = t_1 + t_{17}$	1	41	$t_{74} = t_{42} + t_{73}//y_{11}$	3	70	$t_{109} = t_{44} + t_{108}$	2
13	$t_{44} = t_{12} + t_{30}$	1	42	$t_{75} = t_{71} + t_{73}//y_{23}$	3	71	$t_{110} = t_{67} + t_{109}//y_{28}$	3
14	$t_{45} = t_{13} + t_{31}$	1	43	$t_{76} = t_8 + t_{16}$	1	72	$t_{105} = t_{32} + t_{109}//y_{20}$	3
15	$t_{46} = t_8 + t_{24}$	1	44	$t_{77} = t_{36} + t_{76}//y_{16}$	2	73	$t_{111} = t_{29} + t_7$	1
16	$t_{47} = t_{32} + t_{46}//y_{24}$	2	45	$t_{78} = t_0 + t_{24}$	1	74	$t_{112} = t_{45} + t_{111}$	2
17	$t_{48} = t_9 + t_{25}$	1	46	$t_{79} = t_{52} + t_{78}$	2	75	$t_{113} = t_{73} + t_{112}//y_{29}$	3
18	$t_{49} = t_{33} + t_{48}//y_{25}$	2	47	$t_{80} = t_{77} + t_{79}//y_0$	3	76	$t_{107} = t_{33} + t_{112}//y_{21}$	3
19	$t_{50} = t_{14} + t_{24}$	1	48	$t_{81} = t_{53} + t_{79}//y_{12}$	3	77	$t_{114} = t_{10} + t_{26}$	1
20	$t_{51} = t_{15} + t_{25}$	1	49	$t_{82} = t_8 + t_{34}$	2	78	$t_{115} = t_{32} + t_{34}$	2
21	$t_{52} = t_2 + t_{18}$	1	50	$t_{83} = t_{38} + t_{82}//y_8$	3	79	$t_{116} = t_{114} + t_{115}//y_{26}$	3
22	$t_{53} = t_6 + t_{44}$	2	51	$t_{84} = t_9 + t_{17}$	1	80	$t_{117} = t_{11} + t_{27}$	1
23	$t_{54} = t_3 + t_{19}$	1	52	$t_{85} = t_{37} + t_{84}//y_{17}$	2	81	$t_{118} = t_{33} + t_{35}$	2
24	$t_{55} = t_7 + t_{45}$	2	53	$t_{86} = t_1 + t_{25}$	1	82	$t_{119} = t_{117} + t_{118}//y_{27}$	3
25	$t_{56} = t_2 + t_{32}$	2	54	$t_{87} = t_{54} + t_{86}$	2	83	$t_{120} = t_{35} + t_{41}$	2
26	$t_{57} = t_{39} + t_{56}//y_2$	3	55	$t_{88} = t_{85} + t_{87}//y_1$	3	84	$t_{63} = t_{112} + t_{120}//y_5$	3
27	$t_{58} = t_3 + t_{33}$	2	56	$t_{89} = t_{55} + t_{87}//y_{13}$	3	85	$t_{91} = t_{120} + t_9//y_9$	3
28	$t_{59} = t_{42} + t_{58}//y_3$	3	57	$t_{92} = t_{14} + t_{26}$	1			
29	$t_{60} = t_4 + t_{22}$	1	58	$t_{93} = t_{56} + t_{92}//y_{14}$	3			

niques and propose a new search framework based on them, which can further optimize given circuits. With the low-latency metric, our new framework contributes to many better implementations. It is noted that many heuristics are beneficial from our new techniques and framework. We think that applying these new strategies to other fields is interesting and leave it as promising future work.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper. This work is supported by the National Key Research and Development Program of China (Grant No. 2018YFA0704702), the National Natural Science Foundation of China (Grant No. 62032014), the Major Basic Research Project of Natural Science Foundation of Shandong Province, China (Grant No. ZR202010220025).

References

1. Albrecht, M.R., Driessen, B., Kavun, E.B., Leander, G., Paar, C., Yalçin, T.: Block ciphers - focus on the linear layer (feat. PRIDE). In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference,

Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I. Lecture Notes in Computer Science, vol. 8616, pp. 57–76. Springer (2014), https://doi.org/10.1007/978-3-662-44371-2_4
2. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Camellia: A 128-bit block cipher suitable for multiple platforms - design and analysis. In: Stinson, D.R., Tavares, S.E. (eds.) Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000, Waterloo, Ontario, Canada, August 14-15, 2000, Proceedings. Lecture Notes in Computer Science, vol. 2012, pp. 39–56. Springer (2000), https://doi.org/10.1007/3-540-44983-3_4
3. Avanzi, R.: The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. IACR Trans. Symmetric Cryptol. **2017**(1), 4–44 (2017), https://doi.org/10.13154/tosc.v2017.i1.4-44
4. Baksi, A., Dasu, V.A., Karmakar, B., Chattopadhyay, A., Isobe, T.: Three input exclusive-or gate support for boyar-peralta's algorithm. In: Adhikari, A., Küsters, R., Preneel, B. (eds.) Progress in Cryptology - INDOCRYPT 2021 - 22nd International Conference on Cryptology in India, Jaipur, India, December 12-15, 2021, Proceedings. Lecture Notes in Computer Science, vol. 13143, pp. 141–158. Springer (2021), https://doi.org/10.1007/978-3-030-92518-5_7
5. Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., Regazzoni, F.: Midori: A block cipher for low energy. In: Iwata, T., Cheon, J.H. (eds.) Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9453, pp. 411–436. Springer (2015), https://doi.org/10.1007/978-3-662-48800-3_17
6. Banik, S., Funabiki, Y., Isobe, T.: Further results on efficient implementations of block cipher linear layers. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **104-A**(1), 213–225 (2021), https://doi.org/10.1587/transfun.2020CIP0013
7. Bao, Z., Guo, J., Ling, S., Sasaki, Y.: PEIGEN - a platform for evaluation, implementation, and generation of s-boxes. IACR Trans. Symmetric Cryptol. **2019**(1), 330–394 (2019), https://doi.org/10.13154/tosc.v2019.i1.330-394
8. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9815, pp. 123–153. Springer (2016), https://doi.org/10.1007/978-3-662-53008-5_5
9. Beierle, C., Kranz, T., Leander, G.: Lightweight multiplication in gf(2ˆn) with applications to MDS matrices. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9814, pp. 625–653. Springer (2016), https://doi.org/10.1007/978-3-662-53018-4_23
10. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçin, T.: PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In: Wang, X., Sako, K. (eds.) Advances in Cryp-

tology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7658, pp. 208–225. Springer (2012), https://doi.org/10.1007/978-3-642-34961-4_14
11. Boyar, J., Matthews, P., Peralta, R.: On the shortest linear straight-line program for computing linear forms. In: Ochmanski, E., Tyszkiewicz, J. (eds.) Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings. Lecture Notes in Computer Science, vol. 5162, pp. 168–179. Springer (2008), https://doi.org/10.1007/978-3-540-85238-4_13
12. Boyar, J., Matthews, P., Peralta, R.: Logic minimization techniques with applications to cryptology. J. Cryptol. **26**(2), 280–312 (2013), https://doi.org/10.1007/s00145-012-9124-7
13. Cid, C., Murphy, S., Robshaw, M.J.B.: Small scale variants of the AES. In: Gilbert, H., Handschuh, H. (eds.) Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers. Lecture Notes in Computer Science, vol. 3557, pp. 145–162. Springer (2005), https://doi.org/10.1007/11502760_10
14. Daemen, J., Rijmen, V.: The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition. Information Security and Cryptography, Springer (2020), https://doi.org/10.1007/978-3-662-60769-5
15. Duval, S., Leurent, G.: MDS matrices with lightweight circuits. IACR Trans. Symmetric Cryptol. **2018**(2), 48–78 (2018), https://doi.org/10.13154/tosc.v2018.i2.48-78
16. Jean, J., Nikolić, I., Peyrin, T.: Joltik v1. 3. CAESAR Round **2** (2015)
17. Jean, J., Peyrin, T., Sim, S.M., Tourteaux, J.: Optimizing implementations of lightweight building blocks. IACR Trans. Symmetric Cryptol. **2017**(4), 130–168 (2017), https://doi.org/10.13154/tosc.v2017.i4.130-168
18. Kahn, A.B.: Topological sorting of large networks. Communications of the ACM **5**(11), 558–562 (1962)
19. Kranz, T., Leander, G., Stoffelen, K., Wiemer, F.: Shorter linear straight-line programs for MDS matrices. IACR Trans. Symmetric Cryptol. **2017**(4), 188–211 (2017), https://doi.org/10.13154/tosc.v2017.i4.188-211
20. Leander, G., Moos, T., Moradi, A., Rasoolzadeh, S.: The SPEEDY family of block ciphers engineering an ultra low-latency cipher from gate level for secure processor architectures. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(4), 510–545 (2021), https://doi.org/10.46586/tches.v2021.i4.510-545
21. Li, S., Sun, S., Li, C., Wei, Z., Hu, L.: Constructing low-latency involutory MDS matrices with lightweight circuits. IACR Trans. Symmetric Cryptol. **2019**(1), 84–117 (2019), https://doi.org/10.13154/tosc.v2019.i1.84-117
22. Li, Y., Wang, M.: On the construction of lightweight circulant involutory MDS matrices. In: Peyrin, T. (ed.) Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9783, pp. 121–139. Springer (2016), https://doi.org/10.1007/978-3-662-52993-5_7
23. Lin, D., Xiang, Z., Zeng, X., Zhang, S.: A framework to optimize implementations of matrices. In: Paterson, K.G. (ed.) Topics in Cryptology - CT-RSA 2021 - Cryptographers' Track at the RSA Conference 2021, Virtual Event, May 17-20, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12704, pp. 609–632. Springer (2021), https://doi.org/10.1007/978-3-030-75539-3_25

24. Liu, M., Sim, S.M.: Lightweight MDS generalized circulant matrices. In: Peyrin, T. (ed.) Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9783, pp. 101–120. Springer (2016), https://doi.org/10.1007/978-3-662-52993-5_6
25. Liu, Q., Wang, W., Fan, Y., Wu, L., Sun, L., Wang, M.: Towards low-latency implementation of linear layers. IACR Trans. Symmetric Cryptol. **2022**(1), 158–182 (2022), https://doi.org/10.46586/tosc.v2022.i1.158-182
26. Liu, Q., Wang, W., Sun, L., Fan, Y., Wu, L., Wang, M.: More inputs makes difference: Implementations of linear layers using gates with more than two inputs. IACR Transactions on Symmetric Cryptology **2022**(2), 351–378 (Jun 2022), https://tosc.iacr.org/index.php/ToSC/article/view/9724
27. Maximov, A., Ekdahl, P.: New circuit minimization techniques for smaller and faster AES sboxes. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(4), 91–125 (2019), https://doi.org/10.13154/tches.v2019.i4.91-125
28. Sarkar, S., Syed, H.: Lightweight diffusion layer: Importance of toeplitz matrices. IACR Trans. Symmetric Cryptol. **2016**(1), 95–113 (2016), https://doi.org/10.13154/tosc.v2016.i1.95-113
29. Sim, S.M., Khoo, K., Oggier, F.E., Peyrin, T.: Lightweight MDS involution matrices. In: Leander, G. (ed.) Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9054, pp. 471–493. Springer (2015), https://doi.org/10.1007/978-3-662-48116-5_23
30. Tan, Q.Q., Peyrin, T.: Improved heuristics for short linear programs. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(1), 203–230 (2020), https://doi.org/10.13154/tches.v2020.i1.203-230
31. Xiang, Z., Zeng, X., Lin, D., Bao, Z., Zhang, S.: Optimizing implementations of linear layers. IACR Trans. Symmetric Cryptol. **2020**(2), 120–145 (2020), https://doi.org/10.13154/tosc.v2020.i2.120-145
32. Yang, Y., Zeng, X., Wang, S.: Construction of lightweight involutory MDS matrices. Des. Codes Cryptogr. **89**(7), 1453–1483 (2021), https://doi.org/10.1007/s10623-021-00879-3

Supplementary Materials

A Exhaustive Search Method for SLPD

For computing $\vec{y} = M\vec{t}$, where the matrix M is over $M_{m\times n}$, a circuit is needed with n input signals $\{t_0, t_1, \ldots, t_{n-1}\}$ and m output signals $\{y_0, y_1, \ldots, y_{m-1}\}$. We use the base set $\mathcal{B}$ and the output set $\mathcal{O}$ to represent the input signals and the output signals, respectively. Our goal is to search for a circuit $\mathcal{C}$ from $\mathcal{B}$ to $\mathcal{O}$. $|\cdot|$ is used to denote the number of elements in the set. We have $|\mathcal{B}| = n$ and $|\mathcal{O}| = m$. Additional limitations, such as the minimum depth, are contained in $\mathcal{L}$. In this paper, we always focus on the limitation $\mathcal{D}(\mathcal{C}) = \mathcal{D}_{min}(M)$, which indicates that the circuit $\mathcal{C}$ meets the requirements of SLPD.

A possible solution is the exhaustive search method (see Algorithm 8). For each value y_i in $\mathcal{O}$, we use the circuit set $N_{\mathcal{C}_i}$ to save all the circuits to implement y_i. $N_{\mathcal{B}_i}$ contains the corresponding value sets. Finally, all possible circuits to implement M are saved in $\mathcal{C}_{all}$. However, this is not always practical. In CHES 2019, Maximov and Ekdahl pointed out that the complexity of the exhaustive search is exponential of $|\mathcal{B}|$, and linear of $|\mathcal{O}|$. The exhaustive search can only be readily applied to circuits with no more than ten input values [27].

B Running Time of New Framework

The running time t_{all} consists of three parts. The first part is the number of different combinations of $\mathcal{B}'_{\mathcal{C}}$, $\mathcal{O}'_{\mathcal{C}}$, and $\mathcal{U}'_{\mathcal{C}}$. The second part is the time to run $\mathcal{H}$. The third part is the time to execute MergeCircuit() and Remove(). We define the time to run $\mathcal{H}$ once by $t^{\mathcal{H}}_{\mathcal{B}'_{\mathcal{C}},\mathcal{O}'_{\mathcal{C}}}$ and define the time to execute MergeCircuit() and Remove() by t_{opt}. Then, we discuss how many kinds of combinations of $\mathcal{B}'_{\mathcal{C}}$, $\mathcal{O}'_{\mathcal{C}}$, and $\mathcal{U}'_{\mathcal{C}}$.

All the possible forms of $\mathcal{B}'_{\mathcal{C}}$ can be saved in the set $\mathcal{B}_{all}$, in which

$$\mathcal{B}_{all} = \{\mathcal{B} \cup \mathcal{B}' | \mathcal{B}' \subset \mathcal{T}_{\mathcal{C}}/\mathcal{B}\}.$$

The number of $\mathcal{B}_{all}$ is

$$|\mathcal{B}_{all}| = \binom{|\mathcal{T}_{\mathcal{C}}|-|\mathcal{B}|}{0} + \binom{|\mathcal{T}_{\mathcal{C}}|-|\mathcal{B}|}{1} + \ldots + \binom{|\mathcal{T}_{\mathcal{C}}|-|\mathcal{B}|}{|\mathcal{T}_{\mathcal{C}}|-|\mathcal{B}|-1}.$$

For any $\binom{m}{n}$ $(m \geq n \geq 1)$, we have the following recursion formula:

$$\binom{m}{n} = \binom{m-1}{n-1} + \binom{m-1}{m}.$$

Then, we have

$$\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{n} = 2^n.$$

Thus,

$$|\mathcal{B}_{all}| = 2^{|\mathcal{T}_{\mathcal{C}}|-|\mathcal{B}|} - \binom{|\mathcal{T}_{\mathcal{C}}|-|\mathcal{B}|}{|\mathcal{T}_{\mathcal{C}}|-|\mathcal{B}|} = 2^{|\mathcal{T}_{\mathcal{C}}|-|\mathcal{B}|} - 1.$$

Algorithm 8 Exhaustive Search for SLPD

Input: A matrix M over $M_{m\times n}$
Output: All the circuit $\mathcal{C}$ to implement M

Initial the input set $\mathcal{B}_0 \leftarrow \{t_0, t_1, \ldots . t_{n-1}\}$ and the corresponding circuit $\mathcal{C}_0 \leftarrow \phi$
Initial the output set $\mathcal{O} \leftarrow \{y_0, y_1, \ldots y_{m-1}\}$
$\mathcal{C}_{all} \leftarrow \phi$ ▷ containing all the circuits
for each output signal $y_i \in \mathcal{O}$ **do**
    $N^i_{\mathcal{B}} \leftarrow \phi$ ▷ a set containing all the input sets
    $N^i_{\mathcal{C}} \leftarrow \phi$ ▷ a set containing all the circuits to implement y_i
    Initial $N^i_{\mathcal{B}} \leftarrow \{\mathcal{B}_0\}$ ▷ $N^i_{\mathcal{B}}$ only containing the input set $\mathcal{B}_0$ first
    Initial $N^i_{\mathcal{C}} \leftarrow \{\mathcal{C}_0\}$ ▷ putting the corresponding circuit into $N^i_{\mathcal{C}}$
    FIND = 0
    while FIND = 0 **do**
        for each set $\mathcal{B} \in N^i_{\mathcal{B}}$ and the corresponding circuit $\mathcal{C} \in N^i_{\mathcal{C}}$ **do**
            $N_{\mathcal{B}_{tmp}} = \phi$, $N_{\mathcal{C}_{tmp}} = \phi$ ▷ the sets containing new base sets and circuits
            for each $t_i \in \mathcal{B}$ **do**
                for each $t_j \in \mathcal{B}/\{t_i\}$ **do** ▷ $t_i \neq t_j$
                    $t_{|\mathcal{B}|} = t_i \oplus t_j$
                    if $t_{|\mathcal{B}|} \notin \mathcal{B}$ **then** ▷ generating new value
                        if $t_{|\mathcal{B}|} = y_i$ **then**
                            FIND = 1 ▷ finding the output signal
                        end if
                        $\mathcal{B}' \leftarrow \mathcal{B} \cup \{t_{|\mathcal{B}|}\}$ ▷ generating a new set to contain $t_{|\mathcal{B}|}$
                        $\mathcal{C}' \leftarrow \mathcal{C} \cup \{t_{|\mathcal{B}|} = t_i \oplus t_j\}$ ▷ generating the corresponding circuit
                        $N_{\mathcal{B}_{tmp}} \leftarrow N_{\mathcal{B}_{tmp}} \cup \{\mathcal{B}'\}$ ▷ storing new base set $\mathcal{B}'$
                        $N_{\mathcal{C}_{tmp}} \leftarrow N_{\mathcal{C}_{tmp}} \cup \{\mathcal{C}'\}$ ▷ storing the corresponding circuit
                    end if
                end for
            end for
        end for
        $N^i_{\mathcal{B}} \leftarrow N_{\mathcal{B}_{tmp}}$
        $N^i_{\mathcal{C}} \leftarrow N_{\mathcal{C}_{tmp}}$
    end while
    for each element $\mathcal{B} \in N^i_{\mathcal{B}}$ and the corresponding circuit $\mathcal{C} \in N^i_{\mathcal{C}}$ **do**
        if $y_i \notin \mathcal{B}$ **then**
            Remove $\mathcal{B}$ from $N^i_{\mathcal{B}}$ and remove the corresponding $\mathcal{C}$ from $N^i_{\mathcal{C}}$
        end if
    end for
end for
$\mathcal{C}_{all} = \{c_0 \cup c_1 \ldots \cup c_{m-1} | c_0 \in N^0_{\mathcal{C}}, c_1 \in N^1_{\mathcal{C}}, \ldots c_{m-1} \in N^{m-1}_{\mathcal{C}}\}$
Delete all the circuits which do not meet the depth limitation from $\mathcal{C}_{all}$
Choose the best circuit $\mathcal{C}$ from $\mathcal{C}_{all}$
return $\mathcal{C}$

For the fixed $\mathcal{B}'_{\mathcal{C}}$, all the possible forms of $\mathcal{O}'_{\mathcal{C}}$ are saved in $\mathcal{O}_{all}$, in which

$$\mathcal{O}_{all} = \{\mathcal{O}'_{\mathcal{C}} | \mathcal{O}'_{\mathcal{C}} \subset \mathcal{T}_{\mathcal{C}} / \mathcal{B}'_{\mathcal{C}}\}.$$

Similarly, the number of $\mathcal{O}_{all}$ for the fixed $\mathcal{B}'_{\mathcal{C}}$ is

$$|\mathcal{O}_{all}| = 2^{|\mathcal{T}_{\mathcal{C}}| - |\mathcal{B}'_{\mathcal{C}}|} - 1.$$

As $\mathcal{U}'_{\mathcal{C}} = \mathcal{T}_{\mathcal{C}} / \{\mathcal{B}'_{\mathcal{C}} \cup \mathcal{O}'_{\mathcal{C}}\}$, the total running time t_{all} is

$$t_{all} = \sum_{\mathcal{B}'_{\mathcal{C}} \subset \mathcal{B}_{all}} \sum_{\mathcal{O}'_{\mathcal{C}} \subset \mathcal{O}_{all}} (t^{\mathcal{H}}_{\mathcal{B}'_{\mathcal{C}}, \mathcal{O}'_{\mathcal{C}}} + t_{opt}).$$

C Test for the Matrices from [21]

The results of the matrices from [21] are shown in Table 13. For each Hamming weight, the total number, the number of optimized matrices and the minimum number of `XOR` gates are listed.

Table 13: Test for 4254 MDS matrices in [21].

Hamming weight	**Number**	**Opt.**[a]	**Percentage**[b]	**Max.**[c]	**MinXOR**[d]
148	18	**18**	**100.0%**	3	87
149	48	**48**	**100.0%**	5	87
150	72	**72**	**100.0%**	3	87
151	48	**48**	**100.0%**	3	91
152	60	**57**	**95.0%**	2	91
153	72	**66**	**91.7%**	2	91
154	84	**84**	**100.0%**	3	91
155	24	**17**	**70.8%**	2	95
156	48	**39**	**81.3%**	2	94
157	72	**72**	**100.0%**	2	94
158	84	**84**	**100.0%**	2	95
160	162	**115**	**71.0%**	2	91
161	96	**60**	**62.5%**	2	91
162	132	**94**	**71.2%**	2	91
163	120	**120**	**100.0%**	3	92
164	144	**121**	**84.0%**	3	91
165	240	**219**	**91.3%**	3	88
166	228	**212**	**93.0%**	3	89
167	**216**[e]	**187**	**86.6%**	5	89
168	528	**366**	**69.3%**	8	**85**
169	360	**264**	**73.3%**	3	86
170	432	**317**	**73.4%**	5	87
171	432	**278**	**64.4%**	4	88
172	**534**[e]	**342**	**64.0%**	5	88
All	4254	**3300**	**77.5%**	8	85

[a] The number of matrices that our framework can optimize.
[b] The percentage of matrices that our framework can optimize.
[c] The maximum number of reduced `XOR` gates from our framework.
[d] The minimum number of `XOR` gates.
[e] In [25], there exist two matrices that is counted twice.