Expediting Homomorphic Computation via Multiplicative Complexity-aware Multiplicative Depth Minimization

Mingfei Yu and Giovanni De Micheli

EPFL, Lausanne, Switzerland, mingfei.yu@epfl.ch

Abstract. Fully homomorphic encryption (FHE) enables secure data processing without compromising data access, but its computational cost and slower execution compared to plaintext operations pose challenges. The growing interest in FHE-based secure computation necessitates the acceleration of homomorphic computations. While existing research primarily targets the reduction of the multiplicative depth (MD) of homomorphic circuits, this paper addresses the trade-off between MD reduction and the increase in multiplicative complexity (MC), a critical gap often overlooked during circuit optimization and potentially resulting in suboptimal outcomes. Three contributions are presented: (a) an exact synthesis paradigm for optimal homomorphic circuit implementations, (b) an efficient heuristic algorithm named MC-aware MD minimization, and (c) a homomorphic circuit optimization flow combining MC-aware MD minimization with existing MD reduction techniques. Experimental results demonstrate a 21.32% average reduction in homomorphic computation time and showcase significantly improved efficiency in circuit optimization.

Keywords: Homomorphic Encryption · Multiplicative Depth · Multiplicative Complexity · Logic Synthesis

1 Introduction

First recognized in 1978, the concept of *fully homomorphic encryption* (FHE) refers to a revolutionary encryption paradigm that enables direct computation on ciphertexts without the need for decryption [RAD78]. The groundbreaking *bootstrapping* theorem by Gentry marks the birth of the first HE scheme capable of supporting arbitrary computations[1], known as *leveled FHE*. Whereas continuous developments have significantly improved the practicality of FHE, homomorphic computation is typically orders of magnitude slower than its counterpart on plaintexts. Despite the computational challenges, FHE offers the unique advantage of delegating data processing without compromising data access. This characteristic positions FHE as a promising solution for securing computations, with numerous potential applications in scenarios where privacy is paramount. Examples include outsourcing medical data for diagnosis [CNS+16] and privacy-preserving neural network inference [DSC+19]. The growing interest in FHE-based secure computation techniques underscores the need for continuous efforts to accelerate homomorphic computations.

An HE scheme is considered functionally complete if it supports both additions and multiplications. In leveled FHE schemes, the execution time of each homomorphic operation is intricately tied to the *multiplicative depth* (MD) of the circuit representing the target computation, known as the *homomorphic circuit*. The circuit's MD is defined as the maximum number of sequential homomorphic multiplications in it. Therefore, a significant challenge in accelerating homomorphic computations lies in the quest for lower-MD circuit implementations for the target computation. In particular, there has been considerable research interest in reducing the MD of Boolean circuits [CDS15, CAS17, ACS20, LLOY20]. In this problem, the plaintext space is binary and addition and multiplication on plaintexts correspond to Boolean XOR and AND operations. The extensive attention the problem has received is attributed to the inherent support for high-level programming

language instructions in the binary plaintext space. Such support facilitates a seamless compilation from common programs written in a high-level programming language to their homomorphically encrypted counterparts, ensuring secure execution [CDS15].

The time cost of homomorphic computation is determined by both the MD and the number of the involved homomorphic multiplications, known as the *multiplicative complexity* (MC) of a circuit. Achieving a circuit design with lower MD may come at the expense of a significantly increased MC, potentially prolonging homomorphic computation time. While existing works acknowledge this trade-off, effective optimization algorithms addressing it are lacking due to the challenge of jointly considering both aspects in the optimization process. Consequently, this challenge is often addressed passively by terminating MD reduction techniques when the circuit's MC exceeds a predefined upper bound, leading potentially to sub-optimal designs.

Based on empirical observations, we recognize $\text{MC}\times\text{MD}^2$, termed as *HE cost*, as the cost metric to indicate the acceleration obtained by optimizing a homomorphic circuit. Alongside this proposition, we present three key contributions: (a) an exact synthesis paradigm designed to identify HE cost-optimal circuit implementations for small-scale functions; (b) an efficient heuristic algorithm, named *MC-aware MD minimization*, for finding HE cost-optimal circuit implementations in practical scenarios; and (c) an optimization flow incorporating MC-aware MD minimization and *ESOP balancing* [HS22], the leading MD reduction algorithm, as core components. Experimental results show that the proposed MC-aware MD minimization enables the exploration of a design space considerably different from existing MD reduction techniques. Incorporating MC-aware MD minimization into the optimization flow leads to significant improvements over the state-of-the-art, reducing the homomorphic computation time by an average of 21.32% and achieving orders of magnitude reduction in optimization time.

2 Background

This section provides preliminaries on FHE (Section 2.1) and Boolean circuits (Section 2.2), a summary of existing research on homomorphic computation acceleration via MD reduction (Section 2.3.1), and the motivation of devising MC-aware MD minimization technique (Section 2.3.2).

2.1 (Leveled) Fully Homomorphic Encryption

Modern FHE schemes share a common cryptographic structure. The security is based on the *(ring) learning with error* ((R)LWE) assumption, where *noise* is introduced in a linear system of equations to hide a secret. Although noise ensures security, it accumulates during homomorphic operations. Decryption fails if the accumulated noise in the ciphertext grows beyond a certain threshold. As such schemes only support computations of limited complexity, they are classified as *somewhat homomorphic encryption* (SWHE). The introduction of the *bootstrapping* operation elevated SWHE to the realm of FHE, denoted as *leveled FHE*. Bootstrapping involves the homomorphic decryption of a ciphertext using an encrypted secret key, effectively mitigating the noise in the ciphertext and enabling subsequent computations.

Nevertheless, bootstrapping incurs significant computational costs. In practice, striking a balance between future noise growth and the expense of bootstrapping is crucial. This becomes particularly pertinent given that noise in ciphertexts increases linearly after homomorphic additions but quadratically after homomorphic multiplications. The resulting prohibitive cost of chaining multiplications underscores the necessity of reducing the *multiplicative depth* (MD) of the circuit representing the target computation. Therefore, in leveled FHE schemes such as *BFV* [FV12] and *BGV* [BGV12], the size of ciphertexts and the execution performance of homomorphic operations are heavily influenced by the MD.

The optimization of both circuit MD and the number of bootstrapping operations poses a significant challenge due to its complexity. Previous research has endeavored to address bootstrapping reduction under the assumption of a predefined MD upper bound [LP13, PV15]. Given the complexity of this problem, our study focuses exclusively

on enhancing homomorphic circuit design, with bootstrapping operations falling beyond our scope.

2.2 Boolean circuit

A Boolean circuit can be represented as a logic network, structured as a directed acyclic graph with a node set V and a directed edge set E. The node set V comprises three distinct subsets: a set I of *primary inputs* (PIs) lacking fanin, a set O of *primary outputs* (POs) with singular fanin and lacking fanout, and a set G of logic gates chosen from a library. In FHE schemes with the binary plaintext space configured, multiplications and additions translate to Boolean AND and XOR operations, respectively. Consequently, a homomorphic circuit is represented by a logic network featuring a gate library of 2-input AND and 2-input XOR gates, with optional input negation, commonly referred to as an *XOR-AND graph* (XAG). Due to this correspondence, we use the terms *network* and *circuit* interchangeably.

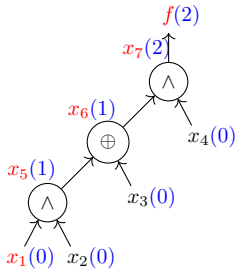

Figure 1: An XAG implementation of function $^{\#}2888a000$. Nodes display the multiplicative levels in blue, with a critical path highlighted in red.

An XAG implementing a 4-variable function, whose truth table is $^{\#}2888a000$[2], is depicted in Fig. 1, where "$\wedge$" and "$\oplus$" denote AND gate and XOR gate, respectively. It consists of PIs $I = \{x_1, x_2, x_3, x_4\}$, POs $O = \{f\}$, gates $G = \{x_5, x_6, x_7\}$, and a set of directed edges E connecting the nodes.

Cut is a concept commonly employed to identify a sub-network or circuit. A cut is characterized by its *root* (a node) and *leaves* (a collection of nodes). A valid set of leaves must satisfy two properties: (1) There is at least one leaf on any path from a PI to the root; (2) All leaves are on at least one such path. A cut is deemed *k-feasible* if its number of leaves does not exceed k, referred to as k-cut. For example, in Fig. 1, there are two 3-cuts rooted at x_7 with leaves $\{x_3, x_4, x_5\}$ and $\{x_4, x_6\}$, respectively. The process of finding all k-cuts in the target network is known as *cut enumeration* [MCB07].

An XAG with $|G|$ gates can be alternatively modeled as a sequence of $|G|$ *steps*, where each step can be represented as

$$x_i = x_{j_1} \circ_i x_{j_2}, \text{ where } \circ_i \in \{\wedge, \oplus\},$$

for $|I| < i \leq |I| + |G|$ and $1 \leq j_1 < j_2 < i$. If an XAG has a single PO, *i.e.*, $|O| = 1$, the function is computed by the last step $x_{|I|+|G|}$. With the *multiplicative level* of each PI defined as 0, the multiplicative level of step x_i, denoted as σ_i, is recursively defined as

$$\sigma_i = \begin{cases} \max\{\sigma_{j_1}, \sigma_{j_2}\} & \text{if } \circ_i = \oplus \\ \max\{\sigma_{j_1}, \sigma_{j_2}\} + 1 & \text{if } \circ_i = \wedge \end{cases} \tag{1}$$

A PO's multiplicative level matches that of its fanin. The MD of an XAG, denoted as d, is defined as the maximal multiplicative level of its steps or gates [CAS17], *i.e.*, $d = \max\{\sigma_i \mid |I| < i \leq |I| + |G|\}$. A path, from a PI to a PO, that determines the MD of an XAG is called the *critical path*, such as the one marked in Fig. 1. The *multiplicative complexity* (MC) of an XAG, denoted as c, is defined as the number of AND gates within it [BPP00], *i.e.*, $c = |\{i \mid \circ_i = \wedge, |I| < i \leq |I| + |G|\}|$.

[2]In this paper, truth tables are represented in hexadecimal as a bit-string, and the most significant bit is on the left-hand side.

2.3 Homomorphic Computation Acceleration

In leveled FHE schemes, the execution of a homomorphic circuit with a larger MD necessitates a larger ciphertext size. This ensures a larger noise budget, allowing computations to proceed without relying on computation-intensive bootstrapping to reduce noise in ciphertexts. As a result, the execution time of a homomorphic circuit correlates with its MD. Consequently, the pursuit of low-MD circuits has been a central focus in existing efforts to accelerate homomorphic computation.

2.3.1 MD Reduction

In the early stages, researchers leveraged circuit depth optimization algorithms within the open-source hardware synthesis tool, ABC [BM10], with the hope that reducing the depth would lead to a decrease in MD [CDS15]. The movement toward customizing optimization algorithms for MD reduction gained momentum with Carpov *et al.*'s seminal work [CAS17]. This work introduced rules for structurally transforming circuits. For example, applying the associativity of the AND operation, $(a \wedge b) \wedge c = a \wedge (b \wedge c)$, to gate a on a critical path could potentially reduce MD by 1 unit. The state-of-the-art homomorphic circuit optimization tool, `LOBSTER` [LLOY20], employs two key steps for MD reduction:(1) offline rule learning, where a set of MD reduction rules is extracted from a training set of circuits, and (2) online term rewriting, where the learned rules are maximally applied to the target circuit to minimize its MD. While `LOBSTER` surpasses previous techniques as it inherently leverages Boolean properties during the learning stage, it does have the drawback of requiring a prohibitively long time for learning and inefficient pattern matching for maximally applying the learned rules.

Although not widely recognized in the cryptographic literature, *ESOP balancing* stands out as the most performant MD reduction technique. It was initially proposed for *T-depth minimization*, a key aspect of *fault-tolerant quantum computing*. Leveraging *exclusive sum-of-products* (ESOPs)' potential as a low-MD XAG, researchers introduced the concept of reducing a circuit's MD by balancing the ESOP representation of each cut [HS22].

2.3.2 Significance of MC-Aware MD Miminization

Achieving a lower MD at the expense of increased MC results in more homomorphic multiplications, impacting overall computation speed. Thus, to optimize homomorphic circuits, MD reduction shall be conducted with MC awareness. This underscores the need for (1) a plausible cost metric reflecting homomorphic circuit execution time, and (2) tailored optimization algorithms supporting such a metric.

Given the substantial time cost of homomorphic multiplication compared to addition, the desired cost metric is the product of the circuit's MC and the complexity of the multiplication corresponding to its MD. However, due to the inherent complexity of HE and the varying characteristics across different schemes, the precise quantitative impact of MD on the speed of homomorphic computation remains an open question. A quantitative link between the MD (d) of a homomorphic circuit and the ciphertext size (l) under the BFV scheme is established by modeling their relationship with a power regression model, expressed as $l = 1.2215 \cdot d^{2.0179}$[CS16]. Additionally, the asymptotic complexity of homomorphic multiplication is recognized as comparable to multiplying arbitrary precision numbers, whose asymptotic runtime bit complexity is known as $O(n \cdot \log(n) \cdot \log(\log(n)))$[ACS20]. Stemming from these empirical observations and for practical simplicity, we introduce $\mathrm{MC} \times \mathrm{MD}^2$, termed as *HE cost*, as a robust metric signifying the execution efficiency of a homomorphic circuit. Our experimental findings in Section 6 validate this metric's efficacy, demonstrating notable acceleration in homomorphic computation time.

3 HE Cost-Optimal Exact Synthesis for Boolean Functions

Within this section, our initial focus centers on tackling the following problem: "Given a Boolean function characterized by limited inputs (no more than 5), how to exactly synthesize its HE cost-optimal circuit implementation?" This dictates the requirement

for an exact synthesis formulation that is simultaneously aware of both the MD and MC of the circuit to be synthesized. Our approach leverages the concepts of *AND fence* and *abstract XAG*.

3.1 Overview of the Methodology

When synthesizing XAGs, a joint application of AND fence and abstract XAG makes it possible to precisely control the usage of AND gates in the objective network [YM23]. We adopt these concepts from their original contexts and subsequently adapt them to enable HE cost-optimal synthesis.

3.1.1 AND Fence

In our definition, an XAG's AND fence, denoted as $\mathcal{F}$, is an ordered set of the number of AND gates in each multiplicative level:

$$\mathcal{F} = \{c_1, c_2, \cdots, c_d\},$$

where d is the MD of the XAG and each c_i within $\mathcal{F}$ is an integer, denoting the number of ANDs within the XAG whose multiplicative level is i. For example, the AND fence of the XAG in Fig. 1 is $\{1, 1\}$. Based on an XAG's AND fence, its MC c can be calculated as:

$$c = \sum_{i=1}^{d} c_i. \tag{2}$$

We attribute the MD and MC to an AND fence $\mathcal{F}$, defining them as the MD and MC of the XAG associated with $\mathcal{F}$. In formal terms, this can be expressed as $MD(\mathcal{F}) = d$ and $MC(\mathcal{F}) = c$. The information within an XAG's AND fence is adequate for deducing both its MC and MD, and therefore, its HE cost, without requiring additional information about node connections.

3.1.2 Abstract XAG

Abstract XAG is a general representation of XAG, in the sense that it removes information regarding the connections among XOR gates in an XAG. This is realized by using *XOR cloud*, an XOR gate whose fanin size is flexible, allowing for any non-zero number of inputs [Soe20], instead of 2-input XOR, as a network component.

In the original definition, an abstract XAG features that:

(a) Each fanin of an AND gate is an XOR cloud;

(b) Each fanin of an XOR cloud is either a PI or an AND gate in a lower *logical* level;

(c) Each PO is an XOR cloud.

In an abstract XAG, each step consists of a 2-input AND gate and its two fanin XOR clouds. Thus, the number of steps in an abstract XAG numerically equals the number of AND gates, which is the MC c of the network. Symbolically, in an abstract XAG that implements an n-variable Boolean function f, each step x_i can be represented as

$$x_i = (\bigoplus_{x_j \in L_{i,1}} x_j) \wedge (\bigoplus_{x_j \in L_{i,2}} x_j),\ n < i \leq n + c,\ 1 \leq j < i, \tag{3}$$

where $L_{i,1}$ and $L_{i,2}$ are respectively the fanins of the two fanin XOR clouds of the AND gate in the i-th step. The PO XOR cloud realizes the function f as a linear function over a set of PIs and steps: denoting L as the fanins of the PO XOR cloud,

$$f = \bigoplus_{x_i \in L} x_i,\ 1 \leq i \leq n + c. \tag{4}$$

Building upon the original definition, we enhance the awareness of multiplicative levels in each step by introducing additional constraints on the feasible fanins of the XOR clouds within a step. This is achieved through overwriting the aforementioned feature (b) with the following two feature descriptions:

i Each fanin of an XOR cloud is a PI or an AND gate in a lower *multiplicative* level;

ii Among the fanins of two XOR clouds connected to an AND gate, at least one fanin should be an AND gate belonging to the preceding multiplicative level.

Through the redefinition, a direct mapping is established between an AND fence $\mathcal{F}$ and the topology of an abstract XAG, thereby extending the definition of an AND fence to an abstract XAG. This correspondence enables any XAG linked to $\mathcal{F}$ to identify its functionally equivalent abstract XAG by configuring the fan-ins of the XOR clouds within the topology, as illustrated in the following example.

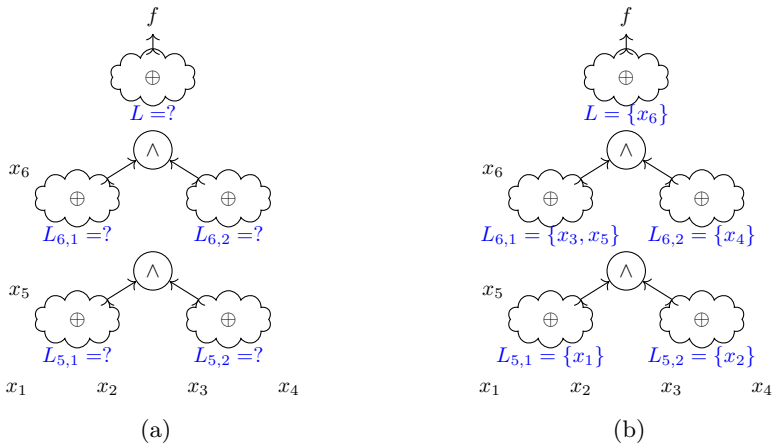

Figure 2: Deterministically deriving the abstract XAG counterpart for the XAG in Fig. 1 in two steps: (a) Determining the topology of the abstract XAG based on the XAG's AND fence; (b) Configuring fanins of each XOR clouds within the abstract XAG according to the node connection in the XAG.

Fig. 2 provides an example illustrating the process of deriving the abstract XAG for an XAG, with the fanins of each XOR cloud explicitly specified beneath it. Given the AND fence of the XAG in Fig. 1, denoted as $\mathcal{F} = \{1, 1\}$, the multiplicative levels of steps x_5 and x_6 are determined to be $\sigma_5 = 1$ and $\sigma_6 = 2$, respectively. This indicates that the topology of the abstract XAG resembles the depiction in Fig. 2a. By configuring the fanins of each XOR cloud correspondingly, an abstract XAG functionally equivalent to the XAG is derived.

Because the abstract XAG counterpart of any XAG with an identical AND fence shares the same topology, an equivalence between the following two tasks emerges: Given a Boolean function f and an AND fence $\mathcal{F}$, (1) determining whether there exists an XAG associated with $\mathcal{F}$ while implementing f, and (2) ascertaining whether there exists a configuration of fanins of the XOR clouds in an abstract XAG topology adhering to $\mathcal{F}$, such that the resulting abstract XAG implements f. The latter can be intuitively formulated as a *Boolean satisfiability* (SAT) problem for efficient solving.

While the conversion of an XAG to an abstract XAG, as depicted in Fig. 2, is deterministic, the reverse process is not. One straightforward approach involves decomposing XOR clouds into 2-input XOR gates, albeit at the cost of introducing more than minimum XOR gates in the resulting XAG. However, since the HE cost of an XAG is determined by its MC and MD — both preserved during such decomposition — if an abstract XAG implementation is HE cost-optimal, its XAG counterpart obtained through XOR cloud decomposition is also HE cost-optimal.

3.1.3 Sketch of Approach

Building upon our redefined concepts of AND fence and abstract XAG, we derive the following insights:

(1) An XAG's HE cost can be derived from its AND fence.

(2) Utilizing SAT solving, we can efficiently determine whether there exists an abstract XAG implementation for a target function while satisfying a specified AND fence.

(3) An abstract XAG can always be converted into an XAG by decomposing XOR clouds, and this process preserves HE cost.

These observations collectively shape our methodology for synthesizing the HE cost-optimal XAG implementation for functions. By enumerating AND fences in an HE cost-ascending manner (leveraging (1)), with each enumeration treated as a SAT problem (drawing on (2)), the HE cost-optimal abstract XAG implementation can be identified, which, in turn, translates to the HE cost-optimal XAG implementation, indicated by (3).

3.2 SAT Encoding

We introduce the requisite variables and clauses, and subsequently encode the previously posited decision problem into a SAT problem.

3.2.1 Variables

Our encoding is based on two kinds of variables: (1) *selection variables*, which encode the fanin configurations of XOR clouds within an abstract XAG, and (2) *function variables*, which encode the function computed at each step.

Depending on whether an XOR cloud is in a step or is the PO, selection variables can be further classified, with each denoted as:

i s_{ijk}, where $n < i \leq n + c$, $1 \leq j < i$, and $k \in \{1, 2\}$, whose *true* value indicates that "step x_j connects to the k-th fanin XOR cloud in step x_i", *i.e.*, $x_j \in L_{i,k}$.

ii s_j, for $1 \leq j \leq n + c$, whose *true* value signifies that "x_j (a PI when $j \leq n$, otherwise a step) is a fanin to the PO XOR cloud".

We denote each function variable as f_{jl}, for $1 \leq j \leq n + c$ and $0 \leq l \leq 2^n$. With $(b_n, \cdots, b_1)_2$ being the binary representation of l, variable f_{jl}'s evaluating to *true* indicates that "PI x_j is assigned to b_j" for $j \leq n$, or "step x_j produces a *true* output given the PI assignment $\{b_1, \cdots, b_n\}$" for $j > n$.

3.2.2 Clauses

Our encoding includes five types of clauses.

For each step x_i, *i.e.*, $n < i \leq n + c$, *clause 1* ensures the correctness of the Boolean operations involved in this step:

$$f_{il} \leftrightarrow \bigwedge_{k \in \{1,2\}} \bigoplus_{j=1}^{i-1} (s_{ijk} \wedge f_{jl}).$$

For each PI x_i, where $1 \leq i \leq n$, $f_{il} = b_i$.

Clause 2 ascertains that the resulting network implements target function f:

$$f(b_1, \cdots, b_n) = \bigoplus_{j=1}^{n+c} (s_j \wedge f_{jl}).$$

In clause 2, the two sides are connected by "=" instead of "↔". This choice is deliberate, as the left-hand side signifies the fixed output of function f under the input assignment $\{b_1,\cdots,b_n\}$, which is a constant rather than a variable.

Clause 3 guarantees that each XOR cloud, whether within a step or serving as the PO, must have a fanin size of at least one:

$$\bigvee_j s_{ijk},\ n < i \leq n+c,\ 1 \leq j < i,\ k \in \{1,2\}$$

and

$$\bigvee_j s_j,\ 1 \leq j \leq n+c.$$

Clause 4 implements feature (b)-i, introduced in Section 3.1.2, as:

$$\bigwedge_{j,k} \overline{s_{ijk}},\ n < i \leq n+c,\ \min\{j' \mid \sigma_{j'} = \sigma_i\} \leq j < i,\ k \in \{1,2\}.$$

Clause 5 incorporates feature (b)-ii as:

$$\bigvee_{j,k} s_{ijk},\ n{+}1 < i \leq n{+}c,\ \min\{j' \mid \sigma_{j'} = (\sigma_i{-}1)\} \leq j \leq \max\{j' \mid \sigma_{j'} = (\sigma_i{-}1)\},\ k \in \{1,2\}.$$

The case $i = n+1$ is excluded from Clause 5 since it is trivially satisfied. Clauses 4 and 5 ensure the abstract XAG topology aligns with AND fence $\mathcal{F}$.

The clauses, if not already in *conjunctive normal form* (CNF), undergo additional conversion into CNF formulas using the Tseitin encoding [Tse83] before being resorted to SAT solving. Using a SAT solver, we can determine whether there exists an abstract XAG with AND fence $\mathcal{F}$ that implements Boolean function f by solving the encoded SAT instance.

3.3 Identification of AND Fence Candidates

In this sub-section, we detail our strategy for identifying which AND fence candidates shall be investigated. This pursuit aligns with our goal sketched in Section 3.1.3 — to ensure the optimality of the synthesized circuit, all AND fences holding the potential shall be considered as candidates. We ascertain the inclusion of the AND fence candidate linked to the HE cost-optimal circuit by accurately determining the ranges, within which its MC and MD may fall. This is realized by first identifying a certain implementation for the target function, whose MC and MD serve as the baseline for looking for its optimal circuit.

When targeting up to 5-variable functions, with a total amount of 2^{32} distinct functions, individually profiling each one of them becomes impractical. Therefore, we employ *affine function classification* [Edw75], a *Boolean classification* technique, to effectively narrow down the problem. Given the invariance of affine-equivalent operations with respect to MC, the MC-minimum XAG implementation of a Boolean function f can be derived from the MC-minimum XAG that represents its affine equivalence class [TSAM19]. Moreover, all 5-variable Boolean functions are categorized into 48 affine equivalence classes [TP15], with the MC-minimum XAG implementation for each representative function already identified in prior research [Soe20]. We further note that the MD of an XAG obtained in this way also remains consistent with its representative XAG, formalized as

Theorem 1. *Affine-equivalent operations are MD-preserving if the multiplicative levels of input variables are identical.*

Proof. The proof of this theorem is provided in Appendix A.3. □

Therefore, for any 5-variable Boolean function f, there always exists an MC-minimum implementation, the MC and MD of which is known, denoted as c_r and d_r. Thus, the HE

cost-optimal implementation of f — unless identical to the MC-minimum one — will have MC and MD values c and d, which satisfy the conditions

$$c \geq c_r,\ d < d_r, \text{ and } c \cdot d^2 < c_r \cdot d_r^2.$$

Given a target MC c and MD d, we denote the complete set of AND fence candidates that satisfy Eq. 2 as $\mathcal{F}(c, d)$. Finding all AND fences $\mathcal{F} \in \mathcal{F}(c, d)$ is essentially a *positive integer partition* problem, a well-studied topic in number theory and combinatorics with detailed solutions in the literature [Knu13]. By considering all AND fences meeting the specified conditions in the synthesis procedure, we ensure minimum HE costs in the synthesized implementations.

3.4 Exact Synthesis Paradigm for Boolean Functions

Algorithm 1: Synthesize Optimal XAGs for Boolean functions

Input: Boolean function f; MC c_r and MD d_r of the MC-minimum XAG for f.
Output: Optimal XAG implementation for f, N.

$d \leftarrow d_r$
known-minimum HE cost $\kappa \leftarrow c_r \cdot d_r^2$
while $d > 1$ **do**
  $d \leftarrow d - 1$
  $c_{max} \leftarrow \lfloor \frac{\kappa}{d^2} \rfloor$
  $keep\_searching \leftarrow false$
  for $c \leftarrow c_r$ **to** c_{max} **do**
    foreach $\mathcal{F} \in \mathcal{F}(c, d)$ **do**
      SAT instance $\zeta \leftarrow SAT\_encoding(f, \mathcal{F})$
      abstract XAG $N' \leftarrow SAT\_solving(\zeta)$
      if $N' \neq NULL$ **then**
        $\kappa \leftarrow c \cdot d^2$
        $keep\_searching \leftarrow true$
        break
    if $keep\_searching$ **then break**
  if ***not*** $keep\_searching$ **then**
    XAG $N \leftarrow decompose\_XOR\_clouds(N')$
    return N

Combining the aforementioned concepts, Algorithm 1 details our paradigm for synthesizing HE cost-optimal XAG implementations for functions. Commencing with the known MC-minimum implementation, we systematically explore the potential existence of an abstract XAG with an MD d one unit smaller than the known solution, denoted as $d = d_r - 1$. Throughout this exploration, we prioritize AND fences with the same MD d in MC-ascending order to consider lower-HE cost candidates first. The exploration extends to AND fences with a further one-unit smaller MD (*i.e.*, decrease d by 1) only if an abstract XAG is successfully synthesized using an AND fence with currently targeted MD d. Termination of the exploration occurs if none of the AND fence candidates with the MD of d leads to a valid implementation. In such cases, it indicates either (a) if $d < d_r - 1$, the previously synthesized abstract XAG, with an MD of $d+1$, is deemed HE cost-optimal; or (b) the known MC-minimum implementation stands HE cost-optimal.

Intuitively, the HE cost should be the sole determinant guiding the investigation of AND fences, irrespective of MD, to prioritize the exploration of lower-HE cost candidates, as sketched in Section 3.1.3. However, Algo. 1 is structured to conduct the exploration based on MD. Specifically, within the set of AND fence candidates whose MD aligns with the currently targeted MD d, Algo. 1 guarantees consideration of the lowest-HE cost one as the initial focus. This strategy is devised based on the observation that aggressively initiating the exploration from the HE cost-minimum AND fence candidate tends to

introduce a considerable number of unsatisfiable SAT instances into the paradigm, leading to an inefficient synthesis procedure. The following observation provides evidence that the strategy does not compromise the optimality of the synthesized XAG.

Theorem 2. *Given a Boolean function f, if there does not exist an abstract XAG implementation for f using an AND fence $\mathcal{F}$ with $MD(\mathcal{F}) = d$, then there is no abstract XAG implementation for f using any AND fence $\mathcal{F}'$ with $MD(\mathcal{F}') = d - 1$.*

Proof. The proof of this theorem is provided in Appendix B. □

4 Exact Synthesis for Sub-circuits

While the exact synthesis approach proposed in Section 3 ensures the optimality of synthesized circuits, scalability concerns limit its application to small-scale functions with a restricted number of inputs. To deliver high-quality solutions for practical functions, we shift our focus from finding the optimal circuit for a *function* to determining the optimal implementation for each *cut*, *i.e.*, sub-circuit, within a baseline circuit implementing the function, ultimately improving the quality of the entire circuit.

4.1 Impacts of Input Levels

The transition in the target problem makes it necessary to explicitly distinguish the concepts of *local multiplicative level* and *global multiplicative level*. When there is no cause for confusion, we omit the term "multiplicative" hereafter. As the name implies, in each sub-circuit highlighted by a cut, the local level of each node, denoted as σ, is computed under the assumption that the leaves operate as PIs, whose levels are regarded as zero. In an XAG implementing such a cut of a circuit, the MD of the XAG precisely corresponds to the local level of the root of the cut. On the other hand, the global level of each node, notated as δ, is calculated concerning the entire circuit. In the problem tackled in the previous section, *i.e.,* "How to exactly synthesize the optimal circuit for a Boolean function," the two concepts are identical. Regarding the problem to address in this section, "How to exactly synthesize the optimal circuit for a cut," the fact that the levels of a cut's leaves are likely non-zero makes the two concepts distinct.

In a n-cut-highlighted sub-circuit, the global levels of the leaves are referred to as the *input level* of the sub-circuit, denoted as

$$\mathcal{L} = \{\delta_1, \delta_2, \cdots, \delta_n\},$$

with δ_i representing the global level of the i-th leaf. $\mathcal{L}$ is *balanced*, if $\delta_1 = \delta_2 = \cdots = \delta_n$; otherwise, it is *imbalanced*. Then, the target problem can be viewed as a generalization of the previous one, where a relaxation that the input level can be imbalanced is introduced.

To ultimately minimize the HE cost of the entire circuit, our goal is to decrease its *MD* by optimizing the implementation of each cut rooted on the critical path. Nevertheless, the presence of non-zero input levels dictates that a lower-MD XAG implementation does not necessarily correlate with a lower global level of the root of the cut implemented by the XAG (*i.e.*, the PO of the XAG), as demonstrated in the following example.

Example 1. The two XAGs depicted in Fig. 3, denoted as N_1 and N_2, both realize a 4-cut $\mathcal{C}$, characterized by the local function f represented in truth table $^{\#}2888a000$. The AND fences associated with N_1 and N_2 are designated as $\mathcal{F}_1 = \{1, 1\}$ and $\mathcal{F}_2 = \{2, 1\}$, respectively. Notably, it is established that N_1 exhibits an MC of 2 and an MD of 2, whereas N_2 is characterized by an MC of 3 and an MD of 2. Undoubtedly, N_1 stands as the more HE cost-efficient implementation of *function* f, given its lower MC. However, a nuanced consideration arises when examining the input level of this cut, denoted as $\mathcal{L} = \{1, 0, 0, 0\}$. From Fig. 3, it is evident that the global depths of the roots (x_7 in Fig. 3a and x_8 in Fig. 3b) are 3 and 2, respectively. This observation points out N_2's potential to serve as a more efficient implementation than N_1 for *cut* $\mathcal{C}$.

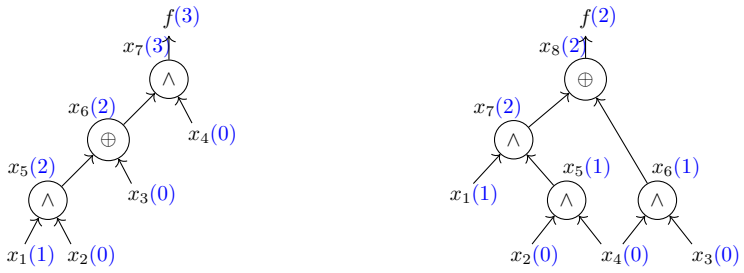

Figure 3: Two XAG implementations for function $^{\#}2888a000$, under the input level $\mathcal{L} = \{1, 0, 0, 0\}$, with each node's global level remarked in blue: (a) XAG N_1; (b) XAG N_2.

To examine the impact of input levels on the divergence between an XAG's MD and the global level of its PO, denoted as δ_{root}, we establish a symbolic analysis. To enhance the intuitiveness of our analysis, we leverage abstract XAG as the logic representation; The conversion from an XAG N to its abstract counterpart N' is deterministic, as depicted in Fig. 2, ensuring generality in our analysis.

Let $\mathcal{C}$ represent the target cut with n leaves and an input level denoted as $\mathcal{L} = \{\delta_1, \cdots, \delta_n\}$. An c-step abstract XAG, whose topology is characterized by an AND fence $\mathcal{F} = \{c_1, \cdots, c_d\}$, is employed to realize the sub-circuit defined by $\mathcal{C}$. The relationship between $\mathcal{F}$ and c adheres to Eq. 2. Given that the PO of an abstract XAG is inherently an XOR cloud, δ_{root} numerically corresponds to the highest global level attained among the c_d steps whose local level σ equals d. This can be symbolically expressed as

$$\begin{aligned} \delta_{root} &= \max\{\delta_i \mid \sigma_i = d, n < i \leq n + c\} \\ &= \max\{\delta_i \mid (n + c - d + 1) \leq i \leq n + c\}. \end{aligned} \tag{5}$$

Denoting the difference between the levels of leaf x_i and the PO XOR cloud as Δ_i, Eq. 5 can be further expressed as:

$$\delta_{root} = \max\{\delta_i + \Delta_i \mid 1 \leq i \leq n\}. \tag{6}$$

It is important to note that $0 \leq \Delta_i \leq d$: The lower bound is reached when a leaf x_i exclusively serves as a fanin of the PO XOR cloud, not contributing to any other XOR clouds within the abstract XAG N'; The upper bound is achieved if a leaf x_i contributes to a step whose local level is 1. While the definition of abstract XAGs guarantees the upper bound, the lower bound is not guaranteed. This observation underscores the strong correlation between δ_{root} and the MD of an (abstract) XAG, denoted as $MD(N)$, when the input level $\mathcal{L}$ is balanced, *i.e.*, when $\delta_1 = \cdots = \delta_n = a$, where a is a positive integer:

$$\begin{aligned} \delta_{root} &= a + d \\ &= a + MD(N) = a + MD(\mathcal{F}). \end{aligned} \tag{7}$$

Examining Eq.7 reveals that when $\mathcal{L}$ is balanced, δ_{root} is collectively influenced by $\mathcal{L}$ and the AND fence $\mathcal{F}$. Conversely, in cases of imbalanced input levels, a meticulous examination of each path becomes essential to identify the critical path determining δ_{root} according to Eq. 6. This necessitates a detailed SAT encoding, extending to the one proposed in Section 3.2.

4.2 Integrating Scheduling into SAT Encoding

To address the challenges posed by imbalanced input levels and ensure the synthesis of optimal implementations under these intricacies, we introduce an extended SAT encoding. In this extended encoding, each SAT instance corresponds not only to a specific AND

fence but also to a particular value assignment for Δ_i for $1 \leq i \leq n$. Following this setup, every SAT instance is linked to a distinct δ_{root}, allowing us to leverage the core concept of Algo. 1 — The iterative solution of a set of SAT instances ensures that the first satisfiable instance corresponds to the optimal implementation of the target cut.

Within the context of an abstract XAG implementation, for a leaf x_i, the statement that "$\Delta_i = a$, where $0 \leq a \leq d$, " is equivalent to asserting "the lowest local level at which x_i contributes to, is $d - a$," where d is the MD of the target AND fence. In simpler terms, determining the value of Δ_i essentially involves scheduling when leaf x_i becomes available to serve as a fanin for a step at that particular local level. Therefore, a specific value assignment to Δ_i for $1 \leq i \leq n$ can be viewed as a *scheduling*, denoted as

$$\mathcal{S} = \{\Delta_1, \Delta_2, \cdots, \Delta_n\}.$$

A scheduling solution can be incorporated into the SAT encoding as an additional clause type concerning the selection variables. By definition, leaf x_j cannot serve as a fanin for any step with a local level lower than $d - \Delta_j$. Therefore, let $x_{i'}$ represent the last step with a local level lower than $d - \Delta_i$, *i.e.*, $\sigma_{i'} = (d - \Delta_i - 1)$ and $\sigma_{i'+1} = (d - \Delta_i)$. The scheduling on leaf x_j is ensured by:

$$\bigwedge_{i,k} \overline{s_{ijk}},\ 1 \leq i \leq i',\ k \in \{1, 2\}.$$

4.3 Strategic Selection of Scheduling Solutions

With the proposed encoding accommodating a scheduling atop a given AND fence, the question arises: which scheduling solutions warrant investigation?

The naïve approach to finding the δ_{root}-minimum implementation for an n-cut $\mathcal{C}$ under a given AND fence $\mathcal{F}$ involves enumerating all $(d+1)^n$ scheduling solutions. However, this exhaustive exploration is inefficient. To address the inefficiency, we propose a strategic selection of promising scheduling solutions, consisting of two key components: (1) Identifying the initial scheduling solution that represents the theoretical minimum δ_{root} under the AND fence $\mathcal{F}$. (2) Identifying the subsequent scheduling solution for exploration, when no feasible implementation exisits under the current scheduling.

Without loss of generality, throughout this section, we assume the input level $\mathcal{L}$ of cut $\mathcal{C}$ satisfies $\delta_1 \leq \cdots \leq \delta_n$ for clarity.

4.3.1 Identification of the Initial Scheduling Solution

Intuitively, the selection of the initial scheduling solution generally aligns with the concept of allowing each leaf to contribute to a step as late as possible. However, we observe that this decision-making process must consider a rule: for each local level, there should be a sufficient number of variables — whether leaves or steps from lower levels — for the steps at the current level to select from as fanins.

Example 2. Consider a scenario where we explore the optimal implementation of a 4-cut $\mathcal{C}$, with local function f and input level $\mathcal{L} = \{0, 0, 0, 1\}$, using an AND fence $\mathcal{F}_1 = \{2\}$. The topology described by $\mathcal{F}_1$ outlines the abstract XAG to be synthesized, featuring two steps, x_5 and x_6, both with local levels $\sigma_5 = \sigma_6 = 1$. Assume the initial scheduling solution is set as $\mathcal{S}_1 = \{1, 1, 0, 0\}$ — meaning steps x_5 and x_6 are restricted to selecting fanins only from leaves x_1 and x_2. This leads to two possible consequences: (1) Both steps x_5 and x_6 implement the same function $x_1 \wedge x_2$. (2) One of the two steps trivially implements a linear function over $\{x_1, x_2\}$. In other words, the satisfiability of the current SAT instance is equivalent to one with the same setup but with an AND fence $\mathcal{F}_2 = \{1\}$. Notably, $\mathcal{F}_2$ exhibits a lower MC compared to $\mathcal{F}_1$, the exploration of which has already been addressed. Hence, the current SAT instance is determined to be unsatisfiable without the need for SAT solving. Judiciously selecting the initial scheduling solution as $\mathcal{S}_2 = \{1, 1, 1, 0\}$ can bypass this trivial case.

Table 1: Number of non-linear functions a step within an Abstract XAG can implement.

#variables	2	3	4	5
#non-linear functions	1	9	55	285

We identify a reasonable initial scheduling solution by addressing the following question: "Given a certain number of variables, how many non-linear functions can a step within an abstract XAG implement?" Recall the general representation of the function implemented by an arbitrary step x_i given by Eq. 3. Our calculation is based on considering all potential configurations of $L_{i,1}$ and $L_{i,2}$, excluding the case where $L_{i,1} = L_{i,2}$, leading to linear functions. We then exclude two cases that result in repetitive functions: (1) Due to the commutativity of the AND operation, functions realized by exchanging $L_{i,1}$ and $L_{i,2}$ are identical; (2) For any function realized with $L_{i,1}$ and $L_{i,2}$, where either $L_{i,1} \subseteq L_{i,2}$ or $L_{i,2} \subseteq L_{i,1}$, there always exists an implementation where $L_{i,1} \not\subseteq L_{i,2}$ or $L_{i,2} \not\subseteq L_{i,1}$ [ÇTP19]. Applying the rules outlined above, we calculate the number of non-linear Boolean functions a step can realize. While only cases with up to 5 variables are summarized in Table 1, the proposed calculations are applicable beyond this limit.

Upon observation, we find that among all AND fence candidates, whose identification is introduced in Section 3.3, the maximum number of steps at the lowest local level does not exceed 15. Interestingly, a step can already realize 55 non-linear functions with 4 variables. This observation implies that, when concentrating on cuts with at most 5 leaves, no AND fence requires initially scheduling all leaves to be available at the lowest local level.

After scheduling the availability of several leaves, ensuring that each step has access to adequate variables, the remaining unscheduled leaves are then scheduled to be available as early as possible, with the condition that they would not form the unique critical path. Assuming leaves x_1 to $x_{j'}$ are already scheduled, the scheduling for an unscheduled leaf x_i, where $j' < i \leq n$, should adhere to the following inequality:

$$\delta_i + \Delta_i \leq \max\{\delta_j + \Delta_j \mid 1 \leq j \leq j'\}.$$

The determination of each Δ_i ensures that the resulting SAT instance inherently covers the exploration space, including cases where leaf x_i is scheduled to a higher level, thereby avoiding repetitive SAT solving.

Algorithm 2: Select initial scheduling for AND fence

Input: AND fence $\mathcal{F} = \{c_1, \cdots, c_d\}$; Input level $\mathcal{L} = \{\delta_1, \cdots, \delta_n\}$.
Output: Scheduling solution $\mathcal{S} = \{\Delta_1, \cdots, \Delta_n\}$; Global level of the root, δ_{root}.

```
1  set of indices of unscheduled leaves s ← {1, ··· , n}
2  number of available variables #vars ← 0
3  for i ← 1 to d do
4  |  number of leaves to schedule #leaves ←look_up(∑_{i'=1}^{i} c_{i'})-#vars
5  |  if #leaves > 0 then
6  |  |  set of indices of leaves to schedule s' ←pop the first #leaves elements in s
7  |  |  foreach j ∈ s' do
8  |  |  |  Δ_j ← i
9  |  |  |  #vars ← #vars + 1
10 |  |  #vars ← #vars + c_i
11 j' ← n − |s|
12 δ_root ← max{δ_j + Δ_j | 1 ≤ j ≤ j'}
13 foreach i ∈ s do
14 |  Δ_i ← δ_root − δ_i
15 return {S, δ_root}
```

The two steps for our initial scheduling solution selection are outlined in Algo. 2. The function *look_up* in Line 4 determines, based on the number of steps, the required number of variables obtained from Table 1. These variables include both leaves scheduled to be available at the current level (Line 8) and steps belonging to lower levels (Line 10).

4.3.2 Identification of the Subsequent Scheduling Solution

When a SAT instance turns out to be unsatisfiable, it implies there does not exist an abstract XAG, whose topology adheres to AND fence $\mathcal{F}$, that implements the target cut $\mathcal{C}$, under the current scheduling solution $\mathcal{S}_1$. Therefore, to find such a $\mathcal{F}$-based implementation, if it does exist, we have to relax the expectation on the resulting level of the root of $\mathcal{C}$, by adjusting $\mathcal{S}_1$ to make the leaves accessible earlier.

If all elements in $\mathcal{S}_1$ are d, indicating all leaves are already scheduled to the lowest level, it means there is no $\mathcal{F}$-based implementation for cut $\mathcal{C}$. Otherwise, each Δ_i in the subsequent scheduling solution $\mathcal{S}_2 = \{\Delta_1, \cdots, \Delta_n\}$ is determined to be $\max\{0, \delta_{root}+1-\delta_i\}$, where δ_{root} is the previously expected level of the root of cut $\mathcal{C}$, calculated with $\mathcal{S}_1$ as the scheduling solution. Given the algorithmic similarity between determining the subsequent scheduling solution and the second step of identifying the initial scheduling solution (Lines 12-14 in Algo. 2), the corresponding pseudocode is omitted.

4.4 Exact Synthesis Paradigm for Sub-circuits

The aforementioned strategies collectively form the foundation of our exact synthesis paradigm for synthesizing optimal implementations for sub-circuits, as outlined in Algo. 3.

Algorithm 3: Synthesize Optimal XAGs for Cuts

Input: Cut $\mathcal{C}$; Current XAG implementation of cut $\mathcal{C}$, N_{old};
Library of AND fence candidates *lib*.
Output: Optimal XAG implementation for $\mathcal{C}$, N.

```
1  N ← exact_synthsis_for_function(C.f)
2  If is_balanced(C.L) then return N
3  δ_root ← calculate_global_level(N,C.L)
4  foreach F ∈ lib do
5  |  {F.S, F.δ_root} ← select_initial_scheduling(F, C.L)
6  target global level at root δ_target ← min{F.δ_root | F ∈ lib}
7  while δ_target < δ_root do
8  |  foreach F ∈ lib do
9  |  |  if F.δ_root = δ_target then
10 |  |  |  SAT instance ζ ← extended_SAT_encoding(C.f,F)
11 |  |  |  abstract XAG N' ← SAT_solving(ζ)
12 |  |  |  if N' ≠ NULL then
13 |  |  |  |  XAG N ← decompose_XOR_clouds(N')
14 |  |  |  |  return N
15 |  |  |  else
16 |  |  |  |  {F.S, F.δ_root} ← update_scheduling(F)
17 |  δ_target ← δ_target + 1
18 return N
```

The exact synthesis paradigm for function (Algorithm 1) is initially invoked to generate a baseline implementation. This implementation is guaranteed to be optimal under the condition of balanced input levels (Line 2). The baseline implementation establishes a known lowest global level at the root of the cut, which serves as an upper bound for the target global level (Line 7). For each AND fence candidate, its initial scheduling solution, along with the corresponding level at the root, is obtained by applying Algorithm 2. The optimality of the solution is ensured by prioritizing the investigation of AND fence with the potential to achieve the lowest global level at the root of the cut. If it turns out to be infeasible, its scheduling solution and the minimum achievable level at the root are updated (Line 16).

4.5 Classifying Exact Synthesis Queries

We address the question: "Under what conditions do two cuts share identical optimal implementations?" Solving this question enables the identification of exact synthesis queries with identical optimal solutions, facilitating the reuse of synthesized solutions and avoiding repetitive SAT solving.

Aligning with our strategy for optimally synthesizing circuits for functions as detailed in Section 3, Boolean classification technique is utilized. However, recognizing that not all affine-equivalent operations preserve MD under imbalanced input levels, we have opted for *NPN classification* [GT62], as NPN-equivalent operations reliably maintain MD (see Appendix A.1 for an in-depth introduction to NPN classification). Through NPN classification, two cuts share the same optimal implementation if: (1) their local functions belong to the same NPN equivalence class, and (2) their input levels align after reordering elements following the *NPN canonicalization* process.

Moreover, we found that the second requirement can be generalized by generating a *signature* for the input level of a cut, facilitating identification based on these signatures. Deriving a signature from the input level of a cut involves two distinct steps.

Lemma 1 (Alignment)**.** *Given two n-cuts $\mathcal{C}_1$ and $\mathcal{C}_2$ with identical local functions and differing input levels $\mathcal{L}_1 = \{\delta_1, \cdots, \delta_n\}$ and $\mathcal{L}_2 = \{\delta'_1, \cdots, \delta'_n\}$, respectively. If, for every $i \in [1, n]$, $\delta_i - \delta'_i = a$ holds, where a is a constant integer, then the optimal implementations of cuts $\mathcal{C}_1$ and $\mathcal{C}_2$ are identical.*

Proof. The proof of this lemma is provided in Appendix C.1. □

Lemma 1 lays the groundwork for the first step of the derivation, termed *alignment*. This step involves subtracting the minimum element δ_1 from each element in $\mathcal{L}$.

Lemma 2 (Dominance)**.** *Let $\mathcal{C}_1$ be an n-cut with input level $\mathcal{L}_1 = \{\delta_1, \cdots, \delta_n\}$ and an optimal implementation with AND fence MD d. If there exists $i \in (1, n]$ such that $\delta_i - \delta_{i-1} \geq d$, then $\mathcal{C}_1$'s optimal implementation is also the optimal implementation of $\mathcal{C}_2$, where $\mathcal{C}_2$ is an n-cut with the same local function as $\mathcal{C}_1$, and its input level is $\mathcal{L}_2 = \{0, \cdots, 0, (\delta_{i+1} - \delta_i), \cdots, (\delta_n - \delta_i)\}$.*

Proof. The proof of this lemma is provided in Appendix C.2. □

Lemma 2 suggests that in the presence of a substantial "gap" among the global levels of the leaves of the target cut, attention should be directed only to those with the potential to form the critical path. The remaining leaves can be treated as if their global levels are zero. This insight gives birth to the *dominance* step.

Algorithm 4: Derive Signature of Input Level

Input: Input level of the target n-cut, $\mathcal{L} = \{\delta_1, \cdots, \delta_n\}$; Threshold θ.
Output: Signature of $\mathcal{L}$, *sig*.

```
1 sig ← L
2 foreach δ ∈ sig do
3 |   δ ← δ − δ_1                                  ▷ Alignment
4 for i ← n to 2 do
5 |   if δ_i − δ_{i−1} ≥ θ then
6 |   |   foreach δ ∈ sig do
7 |   |   |   δ ← max{0, δ − δ_i}                  ▷ Dominance
8 |   |   break
9 return sig
```

The two steps jointly facilitate deriving a signature of the input level of a cut, as outlined in Algo. 4. Especially, when targeting 5-cuts, since the MD of AND fence candidates does not exceed 3, the parameter θ in Algo. 4 is set to 3. The signature of the input level of a cut serves as an additional label of an exact synthesis query, as evidenced by Theorem 3.

Theorem 3. *Given two n-cuts $\mathcal{C}_1$ and $\mathcal{C}_2$ with identical local functions and distinct input levels $\mathcal{L}_1 =$ and $\mathcal{L}_2$. If the signatures derived from $\mathcal{L}_1 =$ and $\mathcal{L}_2$ following Algo. 4 are the same, cuts $\mathcal{C}_1$ and $\mathcal{C}_2$ share the same HE-cost optimal implementation.*

Proof. Follows from Lemmata 1 and 2. □

Effective classification of exact synthesis queries is realized by labelling each query with (1) the NPN representative function of the cut's local function, and (2) the signature of the cut's input level, preventing meaningless invocation of the paradigm (Algo. 3).

5 MC-aware MD Optimization

Building upon Algo. 3, we introduce an innovative MC-aware MD minimization algorithm.Also, we engineer a homomorphic circuit optimization flow. Central to this flow is the MC-aware MD minimization algorithm, complemented by the integration of an advanced MD reduction algorithm, ESOP balancing.

5.1 Overview of the Algorithm

Algorithm 5: MC-aware MD minimization

Input: XAG implementation of the target function, N.
Output: Optimized XAG N_{opt}.

```
1  N_opt ← N
2  cache ← optimal circuits of small-scale functions
3  set of cuts C ← cut_enumeration(N_opt)
4  foreach node n on N_opt's critical path in topological order do
5  |  impl_new ← ∅
6  |  foreach cut C ∈ cuts rooted on n C[n] do
7  |  |  f ← local function of cut C
8  |  |  {representative function f_r, signature sig} ← NPN_classification (f,C.L)
9  |  |  sig ← derive_signature(sig)
10 |  |  impl_new[C] ← cache[{f_r, sig}]
11 |  |  if impl_new[C] = NULL then
12 |  |  |  impl_new[C] ← exact_synthesis_for_cuts(C)
13 |  |  |  cache[{f_r, sig}] ← impl_new[C]
14 |  cut C' ← arg min_{C∈C[n]}(C.δ_n)
15 |  rewrite C' with impl_new[C']
16 return N_opt
```

Algo. 5 outlines the MC-aware MD minimization algorithm. It takes a baseline XAG implementation of the target function as input and outputs one with minimized HE cost.

A cache is utilized to store determined optimal implementations, preventing redundant calls to exact synthesis. As introduced in Section 4.5, NPN-equivalent representative functions and signatures derived from input levels are employed as indices to effectively categorize queries for optimal implementations of cuts.

The algorithm traverses all nodes forming the critical path in topological order (Lines 4). For a given node n, cuts rooted at it are enumerated, and their optimal implementations are obtained and collected in $impl_{new}$. For a cut $\mathcal{C}$ with a local function f and input level $\mathcal{L}$, we determine the NPN-equivalent representative function f_r and derive the signature of $\mathcal{L}$, denoted as sig, which are used to access the cache (Lines 7-8). If the optimal implementation is not yet in the cache, exact synthesis (Algo. 3) is employed, and the cache is subsequently updated (Lines 11-12). Among the optimal implementations of all cuts rooted at node n, the one resulting in the minimum δ_n, *i.e.*, the lowest global level at node n, is selected to replace the original implementation of the corresponding cut in the baseline circuit (Lines 13-14).

5.2 A Homomorphic Circuit Optimization Flow

Intuitively, the proposed MC-aware MD minimization algorithm and ESOP balancing, the most advanced MD optimization algorithm in the literature, explore orthogonal design spaces: In MC-aware MD minimization, the carefully devised AND fence candidate selection scheme excludes those local moves that reduce MD at the cost of a significant increase in MC. In contrast, ESOP balancing aggressively minimizes MD by balancing the ESOP representation of any encountered cut, without any consideration of MC. Their distinct action logics inspire us to build a flow that combines the strengths of both.

Algorithm 6: Homomorphic Circuit Optimization Flow

```
   Input: XAG implementation of the target function, N.
   Output: Optimized XAG N.
   Parameter: num_restarts, cost_metric.
1  N_best ← N, N_opt ← N
2  for i ← 1 to num_restarts do
3  |  N ← N_opt
4  |  if i ≠ 1 then
5  |  |  N ←relaxation(N)
6  |  while true do
7  |  |  optimization algorithm η ← random_engine()
8  |  |  N_opt ← apply_opt_algo(N, η)
9  |  |  if cost_metric(N_opt) ≥ cost_metric(N) then
10 |  |  |  η ← switch_opt_algo(η)
11 |  |  |  N_opt ← apply_opt_algo(N, η)
12 |  |  |  if cost_metric(N_opt) ≥ cost_metric(N) then
13 |  |  |  |  break
14 |  |  |  else
15 |  |  |  |  N ← N_opt
16 |  if cost_metric(N) < cost_metric(N_best) then
17 |  |  N_best ← N
18 return N_best
```

The homomorphic circuit optimization flow is outlined in Algo. 6. The parameter *num_restarts* controls the number of rounds involved in each execution of the flow. In each round, the decision of which optimization algorithm, out of the two candidates, to apply is made by the *random_engine* (Line 7). Once one algorithm is recognized to be unable to further optimize the circuit, *switch_opt_algo* intentionally selects the other algorithm for optimization; If neither of the two algorithms can achieve further optimization, the round is terminated (Lines 9-15). The best design encountered so far is recorded and is regarded as the starting point for the following rounds of exploration. To break out of local optimal, every restart begins with a *relaxation* (Line 5). This procedure significantly increases the MC and MD of the circuit by representing all XOR nodes in the XAG as a combination of three AND nodes, following $a \oplus b = (\overline{a} \wedge b) \vee (a \wedge \overline{b})$ and $a \vee b = \overline{\overline{a} \wedge \overline{b}}$. We also introduce the parameter *cost_metric* to enable reconfigurability of the cost metric in our optimization flow. This setting facilitates straightforward adjustment of the cost metric, providing a foundation for future research within the community to explore metrics that better align with specific FHE schemes and security parameters.

6 Experimental Evaluations

In this section, we showcase our experimental results, organized into two main parts: (1) Evaluation of MC-aware MD minimization as a standalone optimization algorithm. (2) Assessment of the homomorphic circuit optimization flow, which integrates MC-aware MD minimization and ESOP balancing, with MD and HE cost employed as cost metrics.

Table 2: Comparing MC-aware MD minimization against state-of-the-art.

Benchmark	Initial		LOBSTER			ESOP Balancing					MC-aware MD Minimization				
	MC	MD	MC	MD	Exec.[s]	MC	MD	#iter.	Opt.[s]	Exec.[s]	MC	MD	#iter.	Opt.[s]	Exec.[s]
cardio	109	10	116	8	10.00	120	8	3	0.11	10.37	108	8	3	9.81	**9.35**
dsort	708	9	793	8	51.84	948	7	2	0.13	50.49	708	8	2	3.22	**45.18**
msort	810	45	1450	36	**1125.30**	1569	36	8	0.75	1229.30	774	45	2	0.74	1228.88
isort	810	45	1482	36	**1069.88**	1569	36	8	0.75	1227.46	774	45	2	0.49	1212.08
bsort	810	45	1482	36	**1112.19**	1569	36	8	0.75	1226.71	774	45	2	0.49	1213.17
osort	702	25	1404	20	333.97	1404	20	6	0.47	405.50	638	25	2	0.62	**273.56**
hd01	87	6	87	6	5.52	102	5	2	0.00	**3.42**	87	6	1	0.27	5.53
hd02	76	6	76	6	5.24	76	6	1	0.00	5.15	76	6	1	0.27	5.13
hd03	27	5	27	5	1.38	30	4	2	0.02	1.45	29	4	2	0.89	1.43
hd04	75	10	78	8	9.20	74	7	3	0.05	**5.09**	63	8	4	9.53	6.03
hd05	121	7	121	7	8.02	184	6	2	0.01	10.00	121	7	1	0.09	7.95
hd06	121	7	121	7	8.00	184	6	2	0.02	10.02	121	7	1	0.06	8.00
hd07	17	5	13	3	0.99	19	3	2	0.01	0.51	15	3	3	0.78	**0.48**
hd08	18	6	18	5	1.00	26	4	2	0.01	1.24	16	5	2	0.17	**0.94**
hd09	134	14	177	10	20.73	173	10	4	0.06	17.60	134	10	4	8.78	**14.78**
hd10	35	6	36	5	1.71	34	5	1	0.01	1.69	32	5	4	1.51	**1.66**
hd11	391	18	391	15	93.54	411	13	2	0.08	**79.46**	385	14	3	9.74	83.46
hd12	116	16	116	15	26.53	126	12	3	0.09	**20.25**	107	13	3	0.76	24.22
bar	3141	12	3015	11	370.77	2266	8	2	0.15	163.12	1841	9	4	28.85	**149.08**
cavlc	655	16	668	10	63.88	713	8	7	0.37	**52.67**	607	11	7	42.89	76.46
ctrl	107	8	120	5	4.15	107	4	5	0.03	3.97	89	4	5	9.85	**3.49**
dec	304	3	304	3	4.08	304	3	1	0.01	4.13	292	3	2	0.32	**3.97**
i2c	1157	15	1215	8	93.34	1254	7	9	0.24	**71.53**	1152	10	6	19.95	109.30
int2float	213	15	234	8	17.43	240	7	5	0.10	**14.14**	205	10	5	4.34	21.21
router	170	19	190	10	30.18	232	9	5	0.13	**19.34**	186	13	4	1.95	37.28
Total					4468.87					4634.61					4542.62
Norm.					1.00					1.04					1.02

6.1 Experimental Setups

All experiments were conducted on an Apple M1 Max chip with 32GB memory. The following details outline the setups.

Implementation: We implemented the proposed MC-aware MD minimization algorithm using the C++ logic network library `mockturtle`. The exact synthesis solver is based on the C++ reasoning library `bill`, with `glucose` [AS17] chosen as the underneath SAT solver. Both `mockturtle` and `bill` are part of the EPFL logic synthesis libraries [SRHM18]. The back-end executor is built upon the homomorphic encryption library `HElib` [HS20], with the BGV scheme selected and the plaintext space configured to binary. We set the security level to 128-bit(*i.e.*, $\lambda = 128$), and other parameters, within the context of leveled FHE, are configured accordingly by `HElib` to ensure the entire computation can be correctly performed without invoking bootstrapping. To manage the growth of ciphertext size during the computation, we conservatively apply reliearization after each homomorphic AND operation.

Baseline: `LOBSTER` [LLOY20], the state-of-the-art homomorphic circuit optimization tool is chosen as the baseline. Additionally, we introduce an assessment of ESOP balancing [HS22], recognized as the most advanced MD reduction algorithm. This evaluation serves as the first exploration of ESOP balancing's performance in the context of homomorphic circuit optimization. As both ESOP balancing and the proposed MC-aware MD minimization are cut-based, we uniformly set the target cut size as 5.

Benchmark: The 25 involved benchmarks are aligned with [LLOY20]. They are collected from four benchmark suites: `Cingulata` benchmarks[3], homomorphic sorting benchmarks [ÇDSS15], *Hacker's Delight* benchmarks [War13], and EPFL benchmarks [AGM15]. The functions of the benchmarks vary from medical diagnosis, sorting, and bit-twiddling hacks to random/control logic and are believed to represent computations of interest in potential FHE applications. See [LLOY20] for a detailed description of the benchmarks.

6.2 Evaluating MC-aware MD minimization

In Table 2, we report MC, MD, the optimization time in seconds (Opt.)[4], and the circuit execution time in seconds (Exec.). Both ESOP balancing and MC-aware MD minimization are iteratively applied until convergence, and the number of runs is recorded (#iter.). The

[3]Available at: https://github.com/CEA-LIST/Cingulata
[4]Runtime shorter than 0.005s is written as 0.00s

shortest execution time for each benchmark is highlighted in blue. In cases where multiple algorithms produce designs with identical MC and MD values, all of them are highlighted, as the difference in circuit execution times falls within a permissible margin of error. If no improvement over the initial circuit is achieved by any algorithms, no marking is made.

Optimization Time: As MC-aware MD minimization relies on on-the-fly exact synthesis to provide the optimal implementations for the encountered cuts, the optimization time is typically longer than ESOP balancing. However, the optimization time proves to be acceptable. We attribute this achievement to our well-designed SAT encoding, exact synthesis paradigm, and other innovative strategies to avoid unnecessary SAT solving, such as the exact synthesis query classification technique. Due to the unavailability of the source code for `LOBSTER`, we are unable to measure its optimization time. However, based on the information provided in [LLOY20], where a time budget of 125 *hours* is allocated for the rewriting rule learning phase for each benchmark, and the term rewriting phase takes around *eight hours* for relatively large benchmarks such as msort, isort, and bsort, it is reasonable to conclude that MC-aware MD minimization is orders of magnitude faster than `LOBSTER`.

Execution time: Out of the 21 benchmarks where improved designs were generated, `LOBSTER`, ESOP balancing, and MC-aware MD minimization are credited with 3, 8, and 10 designs, respectively. Interestingly, compared to the two involved MD reduction algorithms, MC-aware MD minimization reaches saturation at significantly different design points. Circuits optimized by MC-aware MD minimization consistently exhibit the lowest MC among the three, aligning with the algorithm's philosophy of considering both MC and MD in the optimization process. Additionally, while both ESOP balancing and `LOBSTER` focus exclusively on MD reduction, we observed that ESOP balancing consistently achieves a lower, or at least equal, MD compared to `LOBSTER`.

Challenges in Solo Application of MC-aware MD Minimization: Analysis of cases where the best designs are achieved applying either `LOBSTER` or ESOP balancing reveals instances where MC-aware MD minimization fails to produce low-HE cost designs. This discrepancy may be attributed to the observation that MC-aware MD minimization typically converges with fewer steps compared to ESOP balancing — when exploring the design space, the strict adherence to slightly increased MC may overlook intermediate steps with significantly increased MC, resulting in failure to reach designs with reduced MD. Therefore, a joint application of MC-aware MD minimization and ESOP balancing is proposed, as introduced in Section 5.2, leveraging the latter's ability to reduce MD without increasing MC to complement MC-aware MD minimization, enabling comprehensive exploration of the design space.

6.3 Evaluating Homomorphic Circuit Optimization Flow

In Table 3, the two different settings of the cost metric are distinguished as *HE cost-oriented* and *MD-oriented*. The number of restarts is set to 5 times. The fastest execution achieved on each benchmark, with the results shown in Table 2 considered, is highlighted in blue. Similarly, in cases where the optimization flow produces superior designs with identical MC and MD values under both settings, both circuit execution times are highlighted, despite slight differences. The total execution time is normalized to that achieved by `LOBSTER`.

Effectiveness of the optimization flow: The optimization flow, with either HE cost or MD configured as the cost metric, offers the best implementations for 17 out of the 21 benchmarks where designs better than the initial ones are produced. More remarkably, 11 designs have never been discovered before by solely applying one of the three optimization algorithms, evidencing the flow's ability to comprehensively explore the design space. For instance, on benchmarks msort, isort, and bsort, where `LOBSTER` used to dominate the two algorithms, with HE cost configured as the cost metric, the flow achieved implementations with homomorphic execution respectively 21.68%, 14.43%, and 20.84% faster than those optimized by `LOBSTER`. Notably, in the MD-oriented setting, the flow discovered designs with the new lowest MD values for benchmarks bar, ctrl, and int2float. This also provides convincing evidence that by hybridly applying MC-aware MD minimization and ESOP balancing, the flow can explore the design space that existing algorithms failed to reach.

Table 3: Exploring cost metrics within the proposed flow.

Benchmark	HE cost-oriented				MD-oriented			
	MC	MD	Opt.[s]	Exec.[s]	MC	MD	Opt.[s]	Exec.[s]
cardio	108	8	70.36	**9.35**	117	8	45.61	10.16
dsort	708	7	26.77	**39.42**	948	7	38.28	50.50
msort	788	42	53.86	**881.38**	1391	36	182.86	1180.46
isort	816	42	45.56	**915.46**	1324	36	88.24	1178.53
bsort	788	42	45.45	**880.41**	1354	36	107.31	1214.01
osort	750	24	20.54	320.54	1261	20	121.63	394.29
hd01	102	5	0.14	**3.42**	102	5	0.22	**3.38**
hd02	76	6	3.53	5.13	76	6	1.89	5.15
hd03	30	4	3.88	1.27	30	4	1.46	1.49
hd04	67	7	17.38	**4.81**	74	7	14.05	5.11
hd05	121	7	7.95	8.08	184	6	5.66	10.03
hd06	121	7	5.18	7.99	184	6	7.06	10.00
hd07	15	3	1.02	**0.48**	15	3	0.65	**0.49**
hd08	21	4	0.67	**0.92**	21	4	0.62	**1.12**
hd09	155	10	14.84	16.38	150	10	11.46	16.00
hd10	32	5	1.44	**1.67**	32	5	0.55	**1.71**
hd11	423	13	32.12	83.88	410	13	22.90	80.88
hd12	115	12	3.73	**19.35**	115	12	3.96	**19.30**
bar	1942	8	127.06	**145.10**	2710	7	185.75	160.52
cavlc	691	9	122.10	56.19	717	8	85.62	53.05
ctrl	97	4	10.76	3.71	115	3	12.94	**2.02**
dec	292	3	1.81	**3.96**	292	3	2.28	**3.94**
i2c	1252	7	57.85	**70.74**	1236	8	22.80	91.04
int2float	217	8	32.24	17.24	309	6	10.49	17.10
router	229	9	19.38	**19.06**	257	9	15.03	22.07
Total				3515.94				4532.35
Norm.				0.79				1.01

Impact of cost metric selection: When utilizing HE cost as the cost metric, the optimization flow yields optimal implementations for 15 out of the 21 improved benchmarks, resulting in a 21.32% reduction in homomorphic execution time compared to `LOBSTER`. Conversely, when MD is selected as the cost metric, optimal implementations are achieved for only 7 benchmarks, with a total execution time slightly inferior to `LOBSTER`. It is noteworthy that the MD-oriented flow consistently produces circuits with the lowest MD for nearly all benchmarks, except i2c. Despite its superiority in MD, the resulting designs exhibit inferior execution times, indicating that HE cost is a more suitable cost metric than MD for homomorphic circuit optimization.

7 Conclusions

Existing research in homomorphic computation acceleration via circuit optimization has predominantly focused on reducing the MD of circuits. While MD influences the execution time of individual homomorphic operations, the overall execution time is contingent not only on MD but also on MC. Recognizing this intricate relationship necessitates a more nuanced approach in incorporating the trade-off between MD reduction and MC increase into the homomorphic circuit optimization problem, to achieve enhanced homomorphic computation acceleration. This study is the first to undertake this challenge.

Drawing from empirical observations, we propose HE cost, formulated as $\text{MC}\times\text{MD}^2$, as a refined cost metric to replace MD in homomorphic circuit optimization. To support this metric, which necessitates simultaneous optimization of circuit MC and MD, we introduce (a) an exact algorithm for synthesizing HE cost-optimal circuits, (b) an efficient heuristic algorithm named MC-aware MD minimization, and (c) an optimization flow that integrates our proposal with existing MD reduction techniques from the literature. Experimental evaluations demonstrate that homomorphic circuits optimized by our method achieve an average speedup of 21.32% in execution time, accompanied by substantial reductions in circuit optimization time.

The golden cost metric for homomorphic circuit optimization varies depending on the specific FHE scheme and security parameters, a topic that requires broader exploration. This study offers practical algorithmic tools to tackle this challenge, as both the proposed MC-aware MD minimization algorithm and the circuit optimization flow can be readily adjusted for different cost metrics. This groundwork paves the way for further advancements in homomorphic circuit optimization.

References

[ACS20] Pascal Aubry, Sergiu Carpov, and Renaud Sirdey. Faster homomorphic encryption is not enough: Improved heuristic for multiplicative depth minimization of boolean circuits. In *Proceedings of the Cryptographers' Track at the RSA Conference*, volume 12006, pages 345–363. Springer, 2020.

[AGM15] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. The EPFL combinational benchmark suite. In *Proceedings of International Workshop on Logic & Synthesis*, 2015.

[AS17] Gilles Audemard and Laurent Simon. Glucose SAT solver 4.1, 2017.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science*, pages 309–325. ACM, 2012.

[BM10] Robert K. Brayton and Alan Mishchenko. ABC: an academic industrial-strength verification tool. In *Proceedings of the 22nd International Conference on Computer Aided Verification*, volume 6174, pages 24–40. Springer, 2010.

[BPP00] Joan Boyar, René Peralta, and Denis Pochuev. On the multiplicative complexity of boolean functions over the basis ($\wedge$, $\oplus$, 1). *Theor. Comput. Sci.*, 235(1):43–57, 2000.

[CAS17] Sergiu Carpov, Pascal Aubry, and Renaud Sirdey. A multi-start heuristic for multiplicative depth minimization of boolean circuits. In *Proceedings of the 28th International Workshop on Combinatorial Algorithms*, volume 10765, pages 275–286. Springer, 2017.

[CDS15] Sergiu Carpov, Paul Dubrulle, and Renaud Sirdey. Armadillo: A compilation chain for privacy preserving applications. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing*, pages 13–19. ACM, 2015.

[ÇDSS15] Gizem S. Çetin, Yarkin Doröz, Berk Sunar, and Erkay Savas. Depth optimized efficient homomorphic sorting. In *Proceedings of the 4th International Conference on Cryptology and Information Security in Latin America*, volume 9230, pages 61–80. Springer, 2015.

[CNS+16] Sergiu Carpov, Thanh-Hai Nguyen, Renaud Sirdey, Gianpiero Costantino, and Fabio Martinelli. Practical privacy-preserving medical diagnosis using homomorphic encryption. In *Proceedings of the 9th IEEE International Conference on Cloud Computing*, pages 593–599. IEEE Computer Society, 2016.

[CS16] Ana Costache and Nigel P. Smart. Which ring based somewhat homomorphic encryption scheme is best? In *Proceedings of the Cryptographers' Track at the RSA Conference*, volume 9610, pages 325–340. Springer, 2016.

[ÇTP19] Çagdas Çalik, Meltem Sönmez Turan, and René Peralta. The multiplicative complexity of 6-variable boolean functions. *Cryptogr. Commun.*, 11(1):93–107, 2019.

[DSC+19] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin E. Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. CHET: An optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 142–156. ACM, 2019.

[Edw75] Colin R. Edwards. The application of the rademacher-walsh transform to boolean function classification and threshold logic synthesis. *IEEE Trans. Computers*, 24(1):48–62, 1975.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

[GT62] Eiichi Goto and H. Takahasi. Some theorems useful in threshold logic for enumerating boolean functions. In *Proceedings of the 2nd International Federation for Information Processing Congress*, pages 747–752, 1962.

[HS20] Shai Halevi and Victor Shoup. Design and implementation of helib: a homomorphic encryption library. *IACR Cryptol. ePrint Arch.*, page 1481, 2020.

[HS22] Thomas Häner and Mathias Soeken. Lowering the t-depth of quantum circuits via logic network optimization. *ACM Trans. Quantum Computing*, 3(2), 2022.

[Knu13] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 4: Generating All Trees–History of Combinatorial Generation.* Addison-Wesley Professional, 2013.

[LLOY20] Dongkwon Lee, Woosuk Lee, Hakjoo Oh, and Kwangkeun Yi. Optimizing homomorphic evaluation circuits by program synthesis and term rewriting. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 503–518. ACM, 2020.

[LP13] Tancrède Lepoint and Pascal Paillier. On the minimal number of bootstrappings in homomorphic circuits. In *Financial Cryptography and Data Security*, volume 7862, pages 189–200. Springer, 2013.

[MCB07] Alan Mishchenko, Satrajit Chatterjee, and Robert K. Brayton. Improvements to technology mapping for lut-based fpgas. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 26(2):240–253, 2007.

[PV15] Marie Paindavoine and Bastien Vialla. Minimizing the number of bootstrappings in fully homomorphic encryption. In *Proceedings of the 22nd International Conference on Selected Areas in Cryptography*, volume 9566, pages 25–43. Springer, 2015.

[RAD78] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation*, 4(11):169–180, 1978.

[Soe20] Mathias Soeken. Determining the multiplicative complexity of boolean functions using SAT. *IACR Cryptol. ePrint Arch.*, page 530, 2020.

[SRHM18] Mathias Soeken, Heinz Riener, Winston Haaswijk, and Giovanni De Micheli. The EPFL logic synthesis libraries. *CoRR*, abs/1805.05121, 2018.

[TP15] Meltem Sönmez Turan and René Peralta. The multiplicative complexity of boolean functions on four and five variables. *IACR Cryptol. ePrint Arch.*, page 848, 2015.

[TSAM19] Eleonora Testa, Mathias Soeken, Luca G. Amarù, and Giovanni De Micheli. Reducing the multiplicative complexity in logic networks for cryptography and security applications. In *Proceedings of the 56th Annual Design Automation Conference*, page 74. ACM, 2019.

[Tse83] Grigori S. Tseitin. *On the Complexity of Derivation in Propositional Calculus*, pages 466–483. Springer, 1983.

[War13] Henry S. Warren. *Hacker's Delight, Second Edition.* Pearson Education, 2013.

[YM23] Mingfei Yu and Giovanni De Micheli. Striving for both quality and speed: Logic synthesis for practical garbled circuits. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 1–9. IEEE, 2023.

A Boolean Classification Techniques

Boolean function classification is the process of categorizing Boolean functions into different classes based on various characteristics and properties of the functions. The two Boolean classification techniques employed in this paper are *NPN classification* and *affine function classification*.

A.1 NPN Classification

Based on an n-variable Boolean function $f(x_1, \cdots, x_i, \cdots, x_j, \cdots, x_n)$, the three NPN-equivalent operations are:

1. Input negation: $f \xrightarrow{x_i \to \overline{x_i}} f'$.
2. Input permutation: $f \xrightarrow{x_i \leftrightarrow x_j} f'$.
3. Output negation: $f \xrightarrow{\overline{f}} f'$.

Theorem 4. *NPN-equivalent operations are MD-preserving.*

Proof. Without loss of generality, we consider the impact of input permutation on MD, as it is evident that input and output negations do not affect the MD of an XAG. Recall that the MD d of an XAG with n PIs satisfies

$$d = \max\{\delta_a + \Delta_a \mid 1 \leq a \leq n\}.$$

If we permute any two input variables, say x_i and x_j, the values of δ_i and δ_j exchanged, as are the values of Δ_i and Δ_j. However, this exchange does not affect the value of d. Therefore, NPN-equivalent operations are MD-preserving. □

A.2 Affine Function Classification

Affine function classification is a more effective Boolean function classification technique, in the sense that the set of affine-equivalent operations is the superset of the set of NPN-equivalent operations. Besides the three NPN-equivalent operations, affine-equivalent operations further include:

1. Translational operation: $f \xrightarrow{x_i \to (x_i \oplus x_j)} f'$.
2. Disjoint translational operation: $f \xrightarrow{\oplus x_i} f'$.

Due to the inclusion of the translational operation, affine-equivalent operations are conditionally MD-preserving, as demonstrated in the following sub-section.

A.3 Proof of Theorem 1

Proof. Noting that input negation, input permutation, and output negation have been demonstrated to be MD-preserving operations, without loss of generality, we focus our analysis exclusively on the impact of translational operations and disjoint translational operations on MD.

Recall that the MD d of an XAG with n PIs satisfies

$$d = \max\{\delta_a + \Delta_a \mid 1 \leq a \leq n\}.$$

Considering a translational operation applied to two input variables, x_i and x_j, the term $\delta_i + \Delta_i$ is transformed into $\max\{\delta_i, \delta_j\} + \Delta_i$. This the value of d may change only if:

$$\delta_j > \delta_i, \text{ and } \delta_j + \Delta_i > \max\{\delta_a + \Delta_a \mid 1 \leq a \leq n\}.$$

However, when the multiplicative levels of input variables are identical ($\delta_i = \delta_j$), the aforementioned condition is excluded, demonstrating the MD-preserving nature of translational operations.

For disjoint translational operation on any input variable, x_i, a direct path from x_i to the PO of the XAG is introduced. This operation does not affect the critical paths in the XAG, and consequently, it does not alter the value of d.

Thus, we conclude that affine-equivalent operations are MD-preserving if the multiplicative levels of input variables are identical. □

B Proof of Theorem 2

Proof. Assume, for the sake of contradiction, that there does not exist an abstract XAG implementation for Boolean function f adhering to any AND fence $\mathcal{F}$ such that satisfy $MD(\mathcal{F}) = d$, but there does exist an abstract XAG implementation N for f, whose AND fence $\mathcal{F}_N$ is denoted $\{c_1, \cdots, c_{d-1}\}$.

Upon N, we introduce an additional step x_{c+1}, whose local multiplicative level σ_{c+1} is d. This addition further involves duplicating the original PO XOR cloud of N and configuring them as the two XOR clouds within step x_{c+1}. The fanin of the new PO XOR cloud is exclusively set to x_{c+1}. Terming the new abstract XAG as N', the function implemented by N' is $f \wedge f = f$, and its AND fence $\mathcal{F}_{N'}$ is $\{c_1, \cdots, c_{d-1}, 1\}$, *i.e.*, $MD(N') = d$.

This construction creates a conflict with our initial assumption, leading to the conclusion that there does exist such an abstract XAG implementation N. Therefore, the correctness of the theorem is established. □

C Proof of Signature Derivation Rules

C.1 Proof of Lemma 1

Proof. Assume distinct optimal implementations of cuts $\mathcal{C}_1$ and $\mathcal{C}_2$, denoted as N_1 and N_2 respectively. Let $\delta_{\mathcal{C}_1,N_1}$ and $\delta_{\mathcal{C}_1,N_2}$ represent the global multiplicative levels at the root of $\mathcal{C}_1$, when N_1 and N_2 are respectively applied to cut $\mathcal{C}_1$. Since N_1 is recognized as the optimal implementation for $\mathcal{C}_1$, we have $\delta_{\mathcal{C}_1,N_1} < \delta_{\mathcal{C}_1,N_2}$.

Now, considering the global multiplicative levels at the root of $\mathcal{C}_2$ with N_1 and N_2 respectively applied to $\mathcal{C}_2$, denoted as $\delta_{\mathcal{C}_2,N_1}$ and $\delta_{\mathcal{C}_2,N_2}$, we observe:

$$\delta_{\mathcal{C}_2,N_1} = \delta_{\mathcal{C}_1,N_1} - a < \delta_{\mathcal{C}_1,N_2} - a = \delta_{\mathcal{C}_2,N_2},$$

This inequality implies that N_1 serves as the superior implementation for $\mathcal{C}_2$, leading to a contradiction. Therefore, we conclude the correctness of the lemma. □

C.2 Proof of Lemma 2

Proof. Assume distinct optimal implementations of cuts $\mathcal{C}_1$ and $\mathcal{C}_2$, denoted as N_1 and N_2 respectively. Denote the global multiplicative level at the root of cut $\mathcal{C}_1$, with N_1 applied to cut $\mathcal{C}_1$, as $\delta_{\mathcal{C}_1,N_1}$, then

$$\delta_{\mathcal{C}_1,N_1} = \max\{\delta_a + \Delta_a \mid 1 \le a \le n\}$$

Denote input variable $x_{a'}$ is on the critical path, *i.e.*,

$$\delta_{\mathcal{C}_1,N_1} = \delta_{a'} + \Delta_{a'}.$$

Then, if we apply N_1 to cut $\mathcal{C}_2$, the resulting global multiplicative level at the root of cut $\mathcal{C}_2$, denoted as $\delta_{\mathcal{C}_2,N_1}$, follows

$$\begin{aligned}\delta_{\mathcal{C}_2,N_1} &= \max\{0 + \Delta_1, \cdots, 0 + \Delta_i, (\delta_{i+1} - \delta_i + \Delta_{i+1}), \cdots\} \\ &= \max\{\delta_a - \delta_i + \delta_a \mid 1 \le a \le n\} \\ &= \delta_{a'} - \delta_i + \Delta_{a'}.\end{aligned}$$

Apply N_2 to $\mathcal{C}_2$ and denote the resulting global multiplicative level at the root of $\mathcal{C}_2$, as $\delta_{\mathcal{C}_2,N_2}$. Denote the input variable on the critical path as $x_{b'}$, *i.e.*,

$$\begin{aligned}\delta_{\mathcal{C}_2,N_2} &= \max\{0+\Delta'_1,\cdots,0+\Delta'_i,(\delta_{i+1}-\delta_i+\Delta'_{i+1}),\cdots\}\\ &= \max\{\delta_a-\delta_i+\Delta'_a \mid i+1\le a\le n\}\\ &= \delta_{b'}-\delta_i+\Delta'_{b'}.\end{aligned}$$

Note that a' and b' are not necessarily distinct. Therefore, if we apply N_2 to cut $\mathcal{C}_1$, the resulting global multiplicative level at the root of $\mathcal{C}_1$, denoted as $\delta_{\mathcal{C}_1,N_2}$, shall meet

$$\begin{aligned}\delta_{\mathcal{C}_1,N_2} &= \max\{\delta_a+\Delta'_a \mid 1\le a\le n\}\\ &= \delta_{b'}+\Delta'_{b'}.\end{aligned}$$

Given that N_2 is the optimal implementation of cut $\mathcal{C}_2$, we have

$$\delta_{\mathcal{C}_2,N_2}=\delta_{b'}-\delta_i+\Delta'_{b'}<\delta_{a'}-\delta_i+\Delta_{a'}=\delta_{\mathcal{C}_2,N_1},$$

which suggests that $\delta_{b'}+\Delta'_{b'}<\delta_{a'}+\Delta_{a'}$. Considering the left-hand side and right-hand side of this inequality are respectively the value of $\delta_{\mathcal{C}_1,N_2}$ and $\delta_{\mathcal{C}_1,N_1}$, it implies that N_2 serves as the better implementation for $\mathcal{C}_1$, leading to a contradiction. Therefore, we conclude the correctness of the lemma. □