Faster homomorphic encryption is not enough: improved heuristic for multiplicative depth minimization of Boolean circuits

Pascal Aubry, Sergiu Carpov, and Renaud Sirdey

CEA, LIST,
Point Courrier 172,
91191 Gif-sur-Yvette Cedex, France

Abstract. In somewhat homomorphic encryption schemes (e.g. B/FV, BGV) the size of ciphertexts and the execution performance of homomorphic operations depends heavily on the multiplicative depth. The multiplicative depth is the maximal number of consecutive multiplications for which an homomorphic encryption scheme was parameterized. In this work we propose an improved multiplicative depth minimization heuristic. In particular, a new circuit rewriting operator is introduced, the so called cone rewrite operator. The results we obtain using the new method are relevant in terms of accuracy and performance. Smaller multiplicative depths for a benchmark of Boolean circuits are obtained when compared to a previous work found in the literature. In average, the multiplicative depth is highly improved and the new heuristic execution time is significantly lower. The proposed rewrite operator and heuristic are not limited to Boolean circuits, but can also be used for arithmetic circuits.

Keywords: Somewhat homomorphic encryption · Multiplicative depth · Boolean functions · Heuristic

1 Introduction and related works

We denote by *encryption scheme* the way to encrypt plaintext messages and to decrypt ciphertexts such that discovering the plaintext message from encrypted data is either computationally very hard or even impossible without a secret. An *homomorphic encryption scheme* (HE) allows some operations to be performed directly in the ciphertext space, i.e. without decrypting ciphertexts. An homomorphic encryption is said to be functionally complete when both addition and multiplication operations are supported. Since the seminal work of Gentry [15], many other simpler and more efficient homomorphic encryption schemes have been proposed [5, 6]. A HE scheme with a binary plaintext space allows to execute any Boolean circuit directly over encrypted data.

A common characteristic of HE schemes ciphertexts is the noise component, which is added to the ciphertexts during the encryption for security reasons.

Each homomorphic operation applied on ciphertexts increases this noise component. Decryption correctness cannot be ensured after a predefined number of homomorphic operations as the noise component becomes too large to guarantee exact decryption. Usually, the noise growth induced by the multiplication operation is greater than the noise growth induced by addition. This is why in most cases the *multiplicative depth* of Boolean circuits to be evaluated is considered when HE schemes are parametrized. The multiplicative depth is the maximal number of sequential homomorphic multiplications which can be performed on fresh ciphertexts such that once decrypted we retrieve the result of these multiplications. For an equivalent security level, the increase of circuit multiplicative depth implies larger size ciphertexts and by consequence the cost of homomorphic operations increases also.

Several solutions to ciphertext size increase exist. One of them is the ciphertext bootstrapping procedure introduced in [16] and further developed in [13, 11]. The bootstrapping procedure consists in executing homomorphically the HE scheme decryption algorithm with a noisy ciphertext as input. The noise of the resulting "bootstrapped" ciphertext is lower and independent of the input ciphertext noise. Several works [20, 18, 2] study the problem of minimizing the number bootstrappings in Boolean circuits.

Reducing the multiplicative depth of Boolean circuits is a major impediment in the practical use of somewhat homomorphic encryption. HE scheme parameters increase in size with every multiplicative level. The execution time for the whole Boolean circuit increases accordingly. Many works in the literature treat problems of Boolean circuit optimization for hardware targets or more generally the problem of hardware synthesis. We refer to the open-source software system used for hardware synthesis ABC [3]. It is an open-source environment providing implementations of state-of-the-art circuit optimization algorithms. These algorithms are mainly designed for minimizing circuit area or latency but, currently, none of them is designed for multiplicative depth minimization.

Some of the works in the cryptographic literature [4, 17, 21] focused on the minimization of the number of AND gates in Boolean circuits. [7] deals with the minimization of Boolean circuit depth. This paper presented depth minimization techniques in the context of multi-party computation, with no differentiation between AND and XOR gates.

The authors of the Cingulata toolchain [10] proposed a multi-start priority based heuristic [9] based on multiplicative depth-2 path rewriting operators. These operators decrease locally the multiplicative depth of the circuit. In average, their algorithm managed to lower by more than 3 times the multiplicative depth. Nonetheless, the computational cost of the overall algorithm is very large as the base heuristic is executed several times with different priority functions. None of the proposed priority functions ensures smallest multiplicative depth for all benchmark circuits. Sometimes better results were obtained with a random priority function than with a non-random one.

In this paper, we recall the multiplicative depth-2 path rewrite operator from [9] and generalize it to cone rewriting operators. Afterwards, we present a

new heuristic using the cone rewrite operators. Experimental studies show that smaller multiplicative depth circuits and better computational performances are obtained using the proposed heuristic. We finalize the paper with concluding remarks and some perspectives.

2 Rewrite operators

2.1 Preliminary definitions

We represent a Boolean circuit as a directed acyclic graph $C = (V, E)$ with a set of nodes V and a set of edges E. Circuit nodes represent Boolean functions (gates) and circuit edges are connections between nodes. The set of nodes can be split into 3 independent sub-sets:

- Nodes without a predecessor define circuit inputs. An input can be either a Boolean input variable or a Boolean constant (i.e. logic "0" or logic "1" inputs c_i).
- Nodes without successors (and necessarily with 1 predecessor) define circuit outputs c_o.
- Nodes representing a gate applying a basic Boolean function to the value of its predecessors. The input degree of gates is 2 and the output degree is at least 1. In this work we suppose that the Boolean circuit use AND and XOR operators only. The set {AND,XOR} together with the constant "1" is functionally complete [23]. Every Boolean functions can be expressed by these operators.

Let $\text{pred} : V \to 2^V$ and $\text{succ} : V \to 2^V$ be the functions giving the set of predecessors, respectively successors, of a node $v \in V$ in a Boolean circuit C. We denote $\text{anc} : V \to 2^V$ (resp. $\text{desc} \to 2^V$) the functions giving the set of ancestors (resp. descendants) of a node $v \in V$.

The *multiplicative depth* defines the number of successively executed AND. It influences the parameters of HE schemes. The minimization of the multiplicative depth allows not only to obtain smaller ciphertext sizes but also to minimize the overall execution time of the Boolean circuit. Let us define the function $d : V \to \{0, 1\}$ which return 1 for AND nodes and zero otherwise. The multiplicative depth is influenced only by nodes $v \in V$ such that $d(v) = 1$.

The *multiplicative depth* of nodes is given by $l : V \to \mathbb{N}$. The *multiplicative depth* of a node is the maximum number of AND gates on any path beginning by an input node and ending in node v. l function is defined by:

$$l(v) = \begin{cases} 0 & \text{if } |\text{pred}(v)| = 0, \\ \max_{u \in \text{pred}(v)} l(u) + d(v) & \text{otherwise.} \end{cases}$$

The *reverse multiplicative depth* of nodes is given by $r : V \to \mathbb{N}$. The *reverse multiplicative depth* is the maximum number of AND gates on any path beginning by a successor of v and ending by an output node. r function is defined

by:

$$r(v) = \begin{cases} 0 & \text{if } |\text{succ}(v)| = 0, \\ \max_{u \in \text{succ}(v)} (r(u) + d(u)) & \text{otherwise.} \end{cases}$$

Both l and r are computed recursively. The overall *multiplicative depth* of a circuit C is the maximal multiplicative depth of its nodes:

$$l^{\max} = \max_{v \in V} l(v) = \max_{v \in V} r(v).$$

A node is said *critical* if relation (1) is verified. We denote by *critical circuit* C^* the sub-circuit containing all the critical nodes of circuit C. A *critical path* is a path in this circuit and a *critical cone* is a subset of connected *critical nodes* with a common descendant.

$$l(v) + r(v) = l^{\max},\ v \in V \tag{1}$$

The overall multiplicative depth of circuit C is equal to the multiplicative depth of the critical circuit C^*. Decreasing the multiplicative depth of the *critical circuit* then can be expected to decrease the overall multiplicative depth (and cannot increase it).

2.2 Multiplicative depth-2 path rewriting

In [9] the authors presented two rewriting operators for multiplicative depth minimization. The combined application of these operators allows to reorder circuit gates such that the multiplicative depth is reduced. We improved their method by combining these two operators into a single one. We start by introducing the combined multiplicative depth-2 path rewriting operator and afterwards describe its limitations when applied to arbitrary depth-2 paths.

Let $p = (v_1, U_y, v_t)$ be a path starting and ending with AND gates v_1 and v_t. Between these two gates there is a multi-input XOR[1] gate U_y having inputs v_1 and $y_1, \ldots, y_m$. We denote a_1, a_2 the inputs of node v_1 with $l(a_1) \geq l(a_2)$ and a_3 is the input of v_t other than U_y. Refer to the left-hand side of figure 1 for an illustration. The Boolean formula of path p is $((a_1 \cdot a_2) \oplus \bigoplus_i y_i) \cdot a_3$.

The multiplicative depth-2 path rewrite operator we propose rewrites this path as $((a_2 \cdot a_3) \cdot a_1) \oplus (a_3 \cdot \bigoplus_i y_i)$. Figure 1 illustrates this transformation. Once applied the multiplicative depth *locally* decreases by one (on the path from a_1 to r) if relation (2) is verified.

$$\min_{u \in \text{pred}(v)} l(u) < l(v_1) - 1,\ v \in \{v_1, v_t\}. \tag{2}$$

Entries y_i can be rearranged in a tree structure of 2-input XOR gates after the rewriting in order to obtain again a 2-input gate circuit. Their order does

[1] For the sake of simplicity and without loss of generality, we have grouped intermediary 2-input XOR gates from the initial circuit into a single multi-input XOR gate.

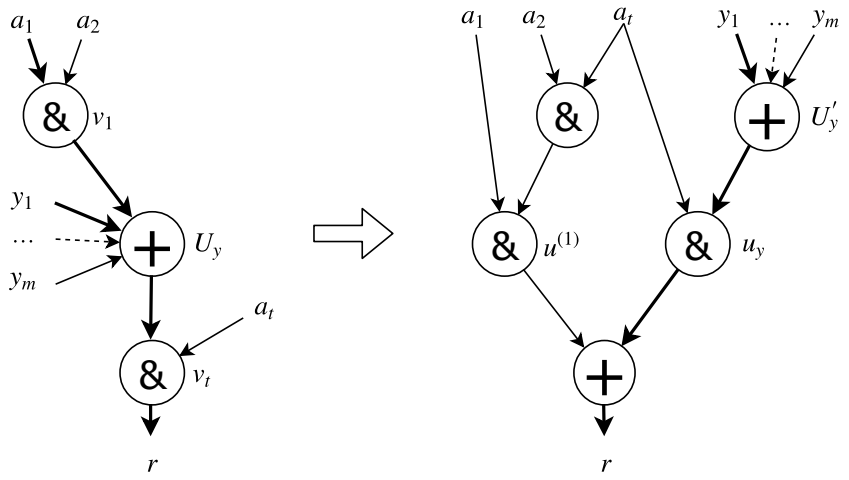

Fig. 1. Rewriting operator for multiplicative depth-2 paths. Bold lines denote critical paths.

not matter. Nonetheless, it would be more interesting to reuse existing XORs for lowering the number of newly created gates. Some special cases need more explanation. If path p does not have any XOR node, then the final path reformulation will be $(a_2 \cdot a_3) \cdot a_1$. If U_y is a 2-input XOR gate, then it will disappear in the transformed circuit and the AND gate u_y will have y_1 and a_3 as inputs.

In the initial path the multiplicative depth of output r is $l(a_1) + 2$. When relation (2) is verified we have $l(a_1) > l(a_2)$ and $l(a_1) > l(a_3)$. After the depth-2 path transformation the multiplicative depth of r becomes:

$$\max(l(a_1), l(y_1), \ldots, l(y_m)) + 1.$$

Suppose that a node y_i, $i = 1, \ldots, m$, is on the critical path before the transformation, i.e. its multiplicative level is $l(y_i) + 1$. After the transformation the multiplicative level of y_i will stay the same, thus the multiplicative level of r does not decrease. *At least* another depth-2 path rewriting on a path ending in u_y is needed in order to decrease the multiplicative depth of r from $l(a_1) + 2$ to $l(a_1) + 1$. For example in the left-hand side of figure 1 if node y_1 is on a critical path then the multiplicative depth of r remains unchanged. We used "at least" previously because a path rewriting will be needed for each input of U_y which belongs to the critical circuit.

The authors of [9] studied only the particular case of multiplicative depth-2 paths where intermediary nodes $y_1, \ldots, y_m$ do not belong to the critical circuit. This limits the applicability of their operator and the number of necessary path rewritings in order to decrease the multiplicative depth.

2.3 Multiplicative depth-2 cone rewriting

We have seen previously that in some cases the overall multiplicative depth does not decrease after a single application of multiplicative depth-2 path operator. In order to address this issue we generalize the multiplicative depth-2 path operator to cones of multiplicative depth 2. We traverse upwards the circuit starting from the sub-set of nodes of $y_1, \ldots, y_m$ which are critical and stop at the first found AND gate. In this way, a cone of multiplicative depth-2 is obtained. In what follows, we introduce a method to rewrite these types of cones such that the overall multiplicative depth decreases.

The cone rewriting operator is equivalent (in terms of multiplicative depth decrease) with the application of the depth-2 path rewriting operator for each critical input of XOR gate U_y (refer to figure 1) as it has been stated earlier. A unified rewrite operator allows to perform a single transformation reducing the multiplicative depth and not several rewrite operator for each critical input of U_y. Also, we seek to reduce the number of newly created nodes after the transformation. The new heuristic we propose is based on that cone rewriting operator. We firstly present the transformation for multiplicative depth-2 critical cones and we further generalize it to cones of arbitrary depth.

A multiplicative depth-2 critical cone δ^2 is a Boolean structure ending by an AND gate v_t and beginning with AND gates $v_1, \ldots, v_n$, such that $v_i \in \mathrm{anc}\,(v_t)$ and $l\,(v_i) = l\,(v_t) - 1$, for any $i = 1, \ldots, n$. The left-hand side of figure 2 illustrates such a cone. The outputs of $v_1, \ldots, v_n$ are combined by a XOR gate U_y (as previously we merged intermediary 2-input XOR gates into one multi-input gate) and connected to one input of node v_t. Let a_t be the input of v_t other than U_y. We denote $a_1^{(i)}$ and $a_2^{(i)}$ the 2 inputs of v_i such that $l\left(a_1^{(i)}\right) \geq l\left(a_2^{(i)}\right)$ and by $y_1, \ldots, y_m$ the inputs of XOR gate U_y which are not critical. By construction we have $l\,(y_i) < l\,(v_t)$ for any y_i.

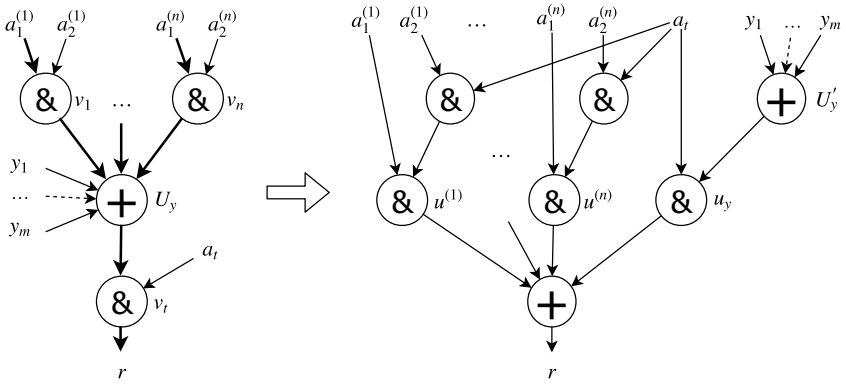

Fig. 2. Rewriting operation for multiplicative depth-2 cone. Bold lines denote critical paths.

Figure 2 illustrates the transformation to be performed in order to decrease the multiplicative depth of cone δ^2. It follows the same idea as the depth-2 path rewriting operator presented earlier. The Boolean formula of the illustrated multiplicative depth-2 cone is:

$$\left(\bigoplus_{i=1}^{n}\left(a_1^{(i)}\cdot a_2^{(i)}\right)\oplus\bigoplus_{i=1}^{m}y_i\right)\cdot a_t.$$

After the rewrite operation the formulation becomes:

$$\left(\bigoplus_{i=1}^{n}\left(a_t\cdot a_2^{(i)}\right)\cdot a_1^{(i)}\right)\oplus\left(a_t\cdot\bigoplus_{i=1}^{m}y_i\right).$$

Thus each AND gate v_i from the input cone is rewritten as $u^{(i)}=\left(a_2^{(i)}\cdot a_t\right)\cdot a_1^{(i)}$ using 2 AND gates and has a smaller multiplicative depth. A new XOR gate $U_y^{'}$ is added for non critical inputs y_i. The output of this gate and a_t are the inputs of a new AND gate u_y. The outputs of gates $u^{(1)},\ldots,u^{(n)}$ and u_y are finally combined together using a multi-input XOR gate. The multiplicative depth of r is reduced by 1 because the following relations are verified (as a consequence of the cone construction procedure):

$$\min_{u\in\mathrm{pred}(v_k)} l\left(u\right)<l\left(v_t\right)-2,\ \forall v_k,k\in\{1,..,n,t\} \tag{3}$$

The main benefit of multiplicative depth-2 cone rewriting is that the multiplicative depth of r is reduced if relations 3 are verified. A single cone transformation is needed instead of n depth-2 path transformations. After cone rewriting only n new AND gates are created (a new gate for each v_i).

2.4 Cone rewriting

Multiplicative depth-2 cone rewrite operators requires that condition (3) is satisfied for all of the cone input nodes, i.e. at least one input of the v_i node must be non critical. In the case when both inputs of v_i are critical we can explore the cones starting with $a_1^{(i)}$ and $a_2^{(i)}$ and build a multiplicative depth-3 cone. If all inputs of only one of these input cones satisfy the reducibility conditions, then the multiplicative depth can be reduced. We can easily extend this operator to multiplicative depths larger than 3.

Our cone construction procedure ConeRec is given in Algorithm 1. It recursively explores the set of critical predecessor nodes starting from node v and incrementally constructs a reducible cone (as the procedure output). If the minimal multiplicative depth to explore is reached (line 2) or at least one predecessor of an AND node v is not critical (line 6) then the exploration stops. Otherwise there are two possibilities as a function of node v type:

AND node If at least one predecessor is reducible then the cone corresponding to this predecessor (or a random one if both are reducible) is added to the result, otherwise the exploration is complete.

Algorithm 1 Recursive algorithm for cone construction.

Require: *minDepth* – explore up to this multiplicative depth

function ConeRec(v) ▷ v – start node
  if $l(v) = minDepth$ **then**
    return $\emptyset$
  end if
  $P \leftarrow \{p \in \text{pred}(v) \mid l(p) = l(v) - d(v)\}$
  if $|P| < 2$ and v is an AND node **then**
    return $\{v\}$
  else
    $\Delta_r \leftarrow \{\text{ConeRec}(p) \mid p \in P\}$
    $\Delta_r \leftarrow \{\delta_r \in \Delta_r \mid \delta_r \neq \emptyset\}$ ▷ reducible input cones
    if v is an AND node **then**
      if $|\Delta_r| = 0$ **then** ▷ no cone is reducible
        return $\emptyset$
      else
        $\delta \leftarrow$ choose randomly from Δ_r
      end if
    else ▷ v is a XOR node
      if $|\Delta_r| = |P|$ **then** ▷ critical cones are reducible
        $\delta \leftarrow \bigcup_{\delta_r \in \Delta_r} \delta_r$
      else
        return $\emptyset$
      end if
    end if
    return $\delta \cup \{v\}$
  end if
end function

XOR node If both predecessors are reducible then the respective cones are added to the result, otherwise exploration is also complete.

To summarize, an AND node is reducible if at least one of its ancestor is reducible and a XOR node is reducible if both its ancestors are reducible.

The ConeRec procedure is called on a circuit node v. If the procedure returns an empty set then the cone ending at v cannot be reduced. Otherwise the procedure output represents the cone to be rewritten and it ensures that the multiplicative depth of this cone can be reduced. We use a $minDepth$ value equal to $l(p) + 1$, where p is the non-critical input of node v.

Observe that the ConeRec procedure when applied to the ending node of a reducible multiplicative depth-2 cone will find exactly that cone. In the case when no reducible multiplicative depth-2 cone ending at v exists the ConeRec procedure will return a cone with a multiplicative depth larger than 2. Rewriting such a cone is very similar to the depth-2 cone rewriting method presented previously. The multiplicative depth cone rewriting is a powerful tool for minimizing the multiplicative depth of Boolean circuits.

In the next section, we introduce the heuristic we have developed to minimize the multiplicative depth of Boolean circuits.

3 Improved heuristic

3.1 Overview

In [9] the authors propose a multi-start heuristic based on multiplicative depth-2 path rewriting operator. This operator is the simplest way to locally reduce the multiplicative depth of a Boolean circuit. Their heuristic use a priority function in order to select the multiplicative depth-2 path to be reduced. None of the priority functions seems to give better results than the others in general as the structure of the Boolean circuit appears to play an important role on which of the priority functions is the most appropriate. Therefore, the authors execute the heuristic with all the priority functions and output the minimal multiplicative depth circuit they obtain. The computational cost of all these executions is therefore high and can be prohibitive for large size Boolean circuits.

The heuristic presented in Algorithm 2 aims at minimizing the multiplicative depth of a given Boolean circuit in a single pass. Indeed, the number of times critical circuits have to be computed is reduced thanks to the proposed cone rewriting operator. At each iteration a set Δ^{min} of cones to minimize is computed. More details about how this set is constructed are given in the next section. If the set Δ^{min} is not empty then the cones from this set are transformed. Afterwards, the multiplicative depths of the circuit nodes are updated. If the multiplicative depth of the new circuit becomes smaller then the output circuit C_{out} is updated. Otherwise, a new set Δ^{min} of cones is computed and the process starts over.

Indeed transforming cones from Δ^{min} does not guarantee that the multiplicative depth is globally reduced as some of the performed reductions may affect

the inputs of critical AND gates. Thus, additional cone transformations may be applicable after this step. The algorithm terminates when the set Δ^{min} is empty. Other termination criteria (e.g. run time, iteration count, multiplicative depth to achieve) can also be considered.

Algorithm 2 Multiplicative depth minimization heuristic based on cone rewriting.

Require: C – input Boolean circuit
Ensure: C_{out} – multiplicative depth optimized
1: $C_{out} \leftarrow C$
2: $\Delta^{min} \leftarrow$ compute reducible cones set
3: **while** Δ^{min} is not empty **do**
4: Rewrite cones from Δ^{min}
5: Update multiplicative depth of C
6: **if** $l_{\max}(C_{out}) > l_{\max}(C)$ **then**
7: $C_{out} \leftarrow C$
8: **end if**
9: $\Delta^{min} \leftarrow$ compute reducible cones set
10: **end while**

3.2 Cone selection method

The goal of the cone selection method is to find a minimal set of cones to rewrite, the reduction of which is likely to lead to a decrease in the overall multiplicative depth. As we have seen earlier any cone rewriting operator adds new nodes to the circuit. So minimizing the set of cones is also beneficial in order to limit the number of newly created nodes.

In order to do so we want to find a minimal size set Δ^{min} of cones such that each critical path in C contains the ending node of at least one cone from this set. Hence, we are guaranteed that the overall multiplicative depth decreases after the cones from Δ^{min} are rewritten. This problem is known as the DVD (DAG vertex deletion) problem [19] in the combinatorial optimization community. The DVD problem is $\mathcal{UG}$-hard [22], thus finding an optimal Δ^{min} in the general case is possible only using an exponential-time algorithm. We propose a heuristic for finding an approximate solution to this problem.

Our cone selection heuristic starts by finding the set Δ of all reducible cones (using the ConeRec procedure). Then, a graph C^{AND} containing the critical AND nodes is built. Two AND nodes are connected in C^{AND} if there is a depth-2 critical path between them in the initial circuit. An AND node is said reducible if it is the (topological) last of a reducible cone. For each cone $\delta \in \Delta$, there is a unique terminal AND node in C^{AND}.

A network-flow inspired Algorithm 3 is used to find a minimal set of cones Δ^{min}. All the nodes v of C^{AND} are visited in topological order. Node flow $f^+(v)$ and edge flow $g^+(v, v)$ are computed for each node and respectively each output

edge of v. Node flow $f^+(v)$ is equal to the sum of flows on input edges or 1 for input nodes. An output edge flow $g^+(v,u)$, $u \in \text{succ}(v)$, is the node flow $f^+(v)$ split equally between node v outputs.

We perform the same computation on graph C^{AND} where the edge directions have been reversed (i.e. in the initial circuit this corresponds to starting from outputs and traversing C^{AND} in reverse topological order) and compute the ascending node flows $f^-(v)$ for each node v. Afterwards, we compute the node weight $f(v)$ defined as the product between its descending and ascending node flows. The node u with the highest weight is selected and deleted from graph C^{AND}. The critical cone terminating in u is added to Δ^{min}. This process (ascending, descending flow computation, etc.) is repeated until C^{AND} is empty.

Finally, the critical cones from Δ^{min} are the ones which are rewritten in Algorithm 2.

Algorithm 3 Cone selection algorithm.

Require: C^{AND} – input circuit
Ensure: Δ^{min} – minimal set of cones
function CompFlow(C) ▷ C – input circuit
  for $v \in C$ in topological order **do**
    if v is input **then**
      $f(v) = 1$
    else
      $f(v) = \sum_{u \in \text{pred}(v)} g(u,v)$
    end if
    $g(v,u) = \frac{f(v)}{|\text{succ}(v)|}$ for all $u \in \text{succ}(v)$
  end for
  return f
end function
$\Delta^{min} \leftarrow \emptyset$
while C^{AND} is not empty **do**
  $f^+ \leftarrow$ CompFlow(C^{AND})
  Reverse circuit C^{AND} edge directions
  $f^- \leftarrow$ CompFlow(C^{AND})
  $f(v) = f^-(v) \cdot f^+(v)$ for all $v \in C^{AND}$
  $u = \arg\max_{v \in C^{AND}} f(v)$
  Remove node u from C^{AND}
  Add critical cone ending at node u to Δ^{min}
end while

3.3 Reductions on non-critical circuits

In some cases, no more reducible cones are available in the critical circuit C^{AND}. Yet, this does not mean that the multiplicative depth of C cannot be further reduced as we could further rewrite non-critical parts of circuit C. This may

decrease the multiplicative depth of certain nodes and, as a consequence, some cones which did not fulfilled the reducibility conditions before may become reducible.

For this purpose, we construct a sub-circuit C_v which contains all the ancestors of a node v. Observe that by computing the critical circuit of C_v and applying Algorithm 2 on this circuit we can reduce the multiplicative depth of v. Afterwards, we verify if there are new reducible cones in C and transform them if this is the case.

In this work, we only reduce sub-circuits C_v such that v is a non-critical input of a critical AND node. We could imagine to extend these reductions to other nodes of C. Still, as we wanted to limit the number of created nodes, we did not explore this idea. Nevertheless, we think it is an interesting perspective for further decreasing the multiplicative depth.

4 Experimental results

We used for our experimentations the Boolean circuits from the EPFL Combinational Benchmark Suite. Three types of combinational circuits are provided: arithmetic, random/control and very large (multi-million gate designs). One can refer to [1] for more details about these benchmarks. In our experiments, only two types of benchmarks are used: 10 arithmetic and 10 random/control circuits. Benchmark circuits have been beforehand optimized and mapped with ABC commands `resyn2` and `map`. `map` command is used to obtain circuit representations with only AND and XOR gates. Table 1 shows the characteristics of the obtained benchmarks after these commands were performed. The same benchmarks were used in [9].

We firstly present results on the minimization of multiplicative depth and afterwards we try to estimate the induced acceleration factor for an homomorphic execution of these circuits.

4.1 Multiplicative depth minimization

The heuristic described in previous section was implemented in C. The binary uses ABC as a helper library. We have executed the new heuristic on a single core of an Intel Core^{TM} i7-7600U CPU @ 2.80GHz. The obtained solutions by the new heuristic and the results from work [9] are shown in Table 2.

The initial characteristics of circuits are also recalled (column "initial"). The notations we use are the multiplicative depth ("×depth"), the number of AND gates ("#AND"), the ratio between the multiplicative depth of the input circuit and the optimized one ("ratio") and the execution time in seconds ("time(s)").

The new heuristic presented in this paper gives better results for almost every circuits in the benchmark. The multiplicative depth is reduced when compared to solutions from [9] for all the arithmetic circuits and lower or equal for all random/control circuits. When the multiplicative depths are equal the number of AND nodes is lower for cavlc, priority and router benchmarks. The voter circuit

Table 1. EPFL Combinational Benchmark Suite characteristics after initial optimization with ABC.

Circuit name	#input	#output	×depth	#AND
adder	256	129	255	509
bar	135	128	12	3141
div	128	128	4253	25219
hyp	256	128	24770	120203
log2	32	32	341	20299
max	512	130	204	2832
multiplier	128	128	254	14389
sin	24	25	161	3699
sqrt	128	64	4968	15571
square	64	128	247	9147
arbiter	256	129	87	11839
ctrl	7	26	8	108
cavlc	10	11	16	658
dec	8	256	3	304
i2c	147	142	15	1161
int2float	11	7	15	213
mem_ctrl	1204	1231	110	44795
priority	128	8	203	676
router	60	30	21	167
voter	1001	1	36	4229

is the only case where the multi-start heuristic [9] gives a better result in terms of AND gate count, although the difference is of only 27 gates. On the other side, the output circuits found by our heuristic for sin and arbiter contain less AND gates and a lower multiplicative depth.

In term of computational performance the new heuristic is clearly faster than the multi-start heuristic and this for example allows to minimize the multiplicative depth of complex circuits such as arbiter, div or sqrt in a reasonable time. For the hyp circuit, the minimal multiplicative depth for the circuits has not been found after 48 hours of execution. Nonetheless, the multiplicative depth has been significantly reduced compared to the multi-start heuristic.

4.2 Homomorphic execution acceleration

In this subsection we study the influence of multiplicative depth minimization on an homomorphic execution of the benchmark circuits. The homomorphic multiplication operation (i.e. the AND gate) is the heaviest one in the somewhat homomorphic encryption schemes described in introduction. We start by explaining how we estimate the complexity of a multiplication operation.

An in-depth study of parameters for homomorphic encryption schemes is performed in [12]. The authors provide in the appendices several samples of HE scheme parameters for different multiplicative depths, plaintext spaces, etc.

Table 2. Solutions obtained by the heuristic proposed in this work (column "this work") and best obtained solutions for the multi-start heuristic [9] (combined priority functions, random and non-random ones). Bold font is used to emphasize the best solutions in terms of multiplicative depth as well as number of AND gates. The ratio between multiplicative depths of the input circuit and the optimized one is shown in columns "ratio".

Circuit	initial		this work				multi-start [9]			
	× depth	#AND	× depth	#AND	ratio	time(s)	× depth	#AND	ratio	time(s)
adder	255	509	**9**	16378	28.3	125	11	1125	23.2	**40.0**
bar	12	3141	**10**	4193	1.2	**0.7**	12	3141	1.0	10.4
div	4253	25219	**532**	190855	8	**3731**	1463	31645	2.9	72000
hyp	24770	120203	**15230**	135433	1.6	172000	24562	120307	1.0	72000
log2	341	20299	**129**	31573	2.6	**94**	141	27362	2.4	14690
max	204	2832	**26**	7666	7.8	**14.5**	27	4660	7.6	1712
multiplier	254	14389	**57**	23059	4.5	**30.73**	59	17942	4.3	14810
sin	161	3699	**74**	**5507**	2.2	**4.5**	76	5922	2.1	652.83
sqrt	4968	15571	**2084**	321555	2.4	107814	4225	18435	1.2	72000
square	247	9147	**26**	11306	9.3	**12.5**	28	10478	8.8	9840
arbiter	87	11839	**10**	**5183**	8.7	**43**	42	8582	2.1	72000
ctrl	8	108	**5**	110	1.6	0.0	**5**	**109**	1.6	0.0
cavlc	16	658	**9**	**667**	1.8	**0.0**	**9**	669	1.8	3.78
dec	3	304	**3**	304	1.0	0.0	**3**	304	1.0	0.0
i2c	15	1161	**7**	1213	2.1	**0.1**	8	1185	1.9	7.26
int2float	15	213	**7**	216	2.1	**0.0**	8	216	1.9	0.24
mem_ctrl	110	44795	**40**	54816	2.4	**85**	45	49175	2.4	66222
priority	203	676	**102**	**876**	2.0	**0.5**	**102**	1106	2.0	22.22
router	21	167	**11**	**198**	1.9	0.0	**11**	204	1.9	0.52
voter	36	4229	**30**	4315	1.2	**1.55**	**30**	**4288**	1.2	112.42

Table 3 shows a sample of parameters for the FV scheme [14] and a Boolean plaintext space [2]. A power regression model is fitted onto this data and used afterwards to extrapolate ciphertext size as a function of multiplicative depth. The power regression model we obtain is $y = 1.2215 \cdot x^{2.0179}$ where x is the multiplicative depth and y is the ciphertext size in kBytes. The obtained model is highly accurate (coefficient of determination > 0.9999 and root mean squared relative error $< 1\%$). Estimated ciphertext sizes are given in the third row of Table 3.

Using this model we are able to estimate the size of ciphertext for a given multiplicative depth. The asymptotic complexity of ciphertext multiplication in HE schemes is comparable to the complexity of multiplying arbitrary-precision numbers. One of the best known algorithms for multiplying arbitrary-precision numbers is the Schönhage–Strassen algorithm. It has an asymptotic run-time bit complexity of $O(n \cdot \log(n) \cdot \log(\log(n)))$. So, to find the run-time complexity of

[2] We have performed the same estimations for other HE schemes (Yashe and BGV) and similar results, as the ones described in what follows, were obtained.

Table 3. Multiplicative depth and ciphertext size (kBytes) for FV scheme instances from [12].

× depth	2	5	10	20	30
size	5	31	127	513	1180
estimated size	4.95	31.4	127.3	515.5	1168.2

multiplying 2 HE ciphertexts we use the ciphertext bit-size as n in the above complexity formula. We consider that the run-time complexity of a Boolean circuit HE execution is equal to the number of AND gates in the circuit scaled by the complexity of ciphertext multiplication at the multiplicative depth of this circuit. For example the run-time complexity of the adder circuit is equal to 509 (number of AND gates) multiplied by $\alpha \cdot \log(\alpha) \cdot \log(\log(\alpha))$, where $\alpha = 8192 \cdot 1.2215 \cdot x^{2.0179}$. Here, the 8192 factor corresponds to the number of bits in 1 kByte (i.e. power regression model units).

We have computed run-time complexities for input circuits, circuits from [9] and circuits generated by the heuristic proposed in this paper. Table 4 shows the ratios between the run-time complexity of optimized circuits and the initial ones. These ratios give an estimation of the acceleration factor between the homomorphic execution of an optimized circuit when compared to the homomorphic execution of the input one. We note that these estimates of the acceleration factor only provide orders of magnitude since other factors (ciphertext key-switch, memory complexity, circuit XOR gates, etc.) which influence the execution time were not considered. Moreover, not considering the size (by consequence memory access times on a real machine) of ciphertexts is advantageous for large ciphertexts (i.e. high multiplicative depth circuits). The third column ("best") provides the best expected run-time acceleration ratio obtained during the execution of the heuristic proposed in this paper. For this purpose, our heuristic returns the circuit with the best run-time complexity instead of the circuit with the lowest depth.

Homomorphic execution times of Boolean circuits depend not only on the multiplicative depth but also on the number of AND gates to be executed. The results presented in Table 4 suggests that circuit optimization heuristics for HE execution should consider other objectives complementary to the multiplicative depth solely. For example, even if the multiplicative depth, 9, of the adder circuit found by our heuristic is smaller compared to the multiplicative depth, 11, of the same circuit from [9] the acceleration factor of our circuit HE execution is 10 times smaller (44.92 vs 419.52). The homomorphic execution of the optimized circuit will be slower than a circuit with a larger multiplicative depth. The best run-time complexity is obtained by our algorithm for a multiplicative depth of 12. Thus, the ratio is below but close the acceleration factor obtained by [9] (408.29 vs 419.52). For several other examples such as sqrt, div or max, the acceleration factor is much higher when choosing the Boolean circuit with the best acceleration ratio.

Table 4. Run-time complexity of optimized circuits compared to initial ones, i.e. how many times faster the execution of an optimized circuit will be.

Circuit	acceleration factor		
	this work		multi-start [9]
	lowest depth	best	
adder	44.92	408.29	**419.52**
bar	**1.17**	**1.17**	1.00
div	10.98	**40.26**	7.66
hyp	**2.47**	**2.47**	1.02
log2	5.19	**5.45**	4.95
max	32.04	**61.03**	48.53
multiplier	15.68	17.46	**18.70**
sin	3.60	**3.80**	3.16
sqrt	0.31	**2.05**	1.19
square	105.81	**109.34**	97.10
arbiter	**257.93**	**257.93**	6.69
ctrl	2.80	2.80	**2.82**
cavlc	**3.51**	**3.51**	3.50
dec	1.00	1.00	1.00
i2c	**5.16**	**5.16**	3.93
int2float	3.93	3.93	3.95
mem_ctrl	**7.43**	**7.43**	6.32
priority	**3.40**	**3.40**	2.69
router	**3.50**	**3.50**	3.40
voter	1.47	1.47	1.47

5 Conclusion and perspectives

In this work, we proposed an improved method for minimizing the multiplicative depth of Boolean circuits. In order to do so, we introduced new advanced operators for rewriting critical paths and cones. The presented heuristic is based on these rewriting operators. The multiplicative depth of Boolean circuits is reduced by searching for a set of reducible cones and then rewriting them. This heuristic gives better results compared to the method from [9] in terms of multiplicative depth and HE execution time. For a majority of benchmarks we have obtained smaller multiplicative depth circuits within a much smaller computational budget. We also experimentally demonstrated that, in the context of an homomorphic execution of Boolean circuits, the minimization of multiplicative depth is beneficial only if the number of newly created AND gates is below a threshold.

Some further improvements of the heuristic are envisaged as perspectives. For example, the trade-off between reduction of multiplicative depth and the number of newly created AND gates must be made more precise in the context of HE execution. An interesting approach would be to determine a budget of AND gates to be created at each iteration of our algorithm. Indeed, we can compute the cost, in terms of number of newly added AND gates, of a cone transformation before performing it. Another approach would be to try to minimize the number of AND gates between two iteration of our heuristic.

In the literature can be found some HE implementations of algorithms with a low multiplicative depth and small number of AND gates but with a huge amount of XOR gates [8]. In such kind of circuits, the computational time and the influence on ciphertext noise of XOR gates must be taken into account too. An interesting perspective would be to measure the noise increase incurred by the homomorphic execution of a Boolean circuit, and, to propose heuristics which try to optimize this noise instead of the multiplicative depth (or an estimation of the acceleration factor).

References

1. Amarú, L., Gaillardon, P.E., De Micheli, G.: The EPFL Combinational Benchmark Suite. In: Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS) (2015)
2. Benhamouda, F., Lepoint, T., Mathieu, C., Zhou, H.: Optimization of Bootstrapping in Circuits. In: SODA. pp. 2423–2433. SIAM (2017)
3. Berkeley Logic Synthesis and Verification Group: ABC: A System for Sequential Synthesis and Verification, Release 30308. http://www.eecs.berkeley.edu/~alanmi/abc/, http://www.eecs.berkeley.edu
4. Boyar, J., Peralta, R.: Concrete Multiplicative Complexity of Symmetric Functions. In: MFCS. Lecture Notes in Computer Science, vol. 4162, pp. 179–189 (2006)
5. Brakerski, Z.: Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In: Advances in Cryptology - Crypto 2012. Lecture Notes in Computer Science, vol. 7417, pp. 868–886. Springer (2012)

6. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. pp. 309–325. ITCS '12 (2012)
7. Buescher, N., Holzer, A., Weber, A., Katzenbeisser, S.: Compiling Low Depth Circuits for Practical Secure Computation. In: European Symposium on Research in Computer Security. pp. 80–98. Springer (2016)
8. Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., Sirdey, R.: Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. Journal of Cryptology **31**(3), 885–916 (2018)
9. Carpov, S., Aubry, P., Sirdey, R.: A multi-start heuristic for multiplicative depth minimization of boolean circuits. In: International Workshop on Combinatorial Algorithms. pp. 275–286. Springer (2017)
10. Carpov, S., Dubrulle, P., Sirdey, R.: Armadillo: A Compilation Chain for Privacy Preserving Applications. In: Proceedings of the 3rd International Workshop on Security in Cloud Computing. pp. 13–19. SCC '15 (2015)
11. Chillotti, I., Gama, N., Georgieva, M., Izabachene, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 3–33. Springer (2016)
12. Costache, A., Smart, N.P.: Which ring based somewhat homomorphic encryption scheme is best? In: Cryptographers' Track at the RSA Conference. pp. 325–340. Springer (2016)
13. Ducas, L., Micciancio, D.: Fhew: bootstrapping homomorphic encryption in less than a second. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 617–640. Springer (2015)
14. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptology ePrint Archive **2012**, 144 (2012)
15. Gentry, C.: Fully Homomorphic Encryption Using Ideal Lattices. In: Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing. pp. 169–178. STOC '09 (2009)
16. Gentry, C., Halevi, S., Smart, N.P.: Better Bootstrapping in Fully Homomorphic Encryption. In: Fischlin, M., Buchmann, J.A., Manulis, M. (eds.) Public Key Cryptography. Lecture Notes in Computer Science, vol. 7293, pp. 1–16 (2012)
17. Kolesnikov, V., Sadeghi, A.R., Schneider, T.: Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. In: CANS. Lecture Notes in Computer Science, vol. 5888, pp. 1–20 (2009)
18. Lepoint, T., Paillier, P.: On the Minimal Number of Bootstrappings in Homomorphic Circuits. In: Financial Cryptography Workshops. Lecture Notes in Computer Science, vol. 7862, pp. 189–200 (2013)
19. Paik, D., Reddy, S., Sahni, S.: Deleting vertices to bound path length. IEEE transactions on computers (9), 1091–1096 (1994)
20. Paindavoine, M., Vialla, B.: Minimizing the Number of Bootstrappings in Fully Homomorphic Encryption. In: SAC. Lecture Notes in Computer Science, vol. 9566, pp. 25–43 (2015)
21. Schneider, T., Zohner, M.: GMW vs. Yao? Efficient Secure Two-Party Computation with Low Depth Circuits. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 7859, pp. 275–292 (2013)
22. Svensson, O.: Hardness of vertex deletion and project scheduling. In: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, pp. 301–312. Springer (2012)

23. Wernick, W.: Complete sets of logical functions. Transactions of the American Mathematical Society **51**(1), 117–132 (1942)