Lowering the $T$-depth of Quantum Circuits By Reducing the Multiplicative Depth Of Logic Networks

Thomas Haener  
Mathias Soeken
Microsoft Quantum, Switzerland

ABSTRACT
The multiplicative depth of a logic network over the gate basis \{\land, \lor, \neg\} is the largest number of \land gates on any path from a primary input to a primary output in the network. We describe a dynamic programming based logic synthesis algorithm to reduce the multiplicative depth in logic networks. It makes use of cut enumeration, tree balancing, and exclusive sum-of-products (ESOP) representations. Our algorithm has applications to cryptography and quantum computing, as a reduction in the multiplicative depth directly translates to a lower $T$-depth of the corresponding quantum circuit. Our experimental results show improvements in $T$-depth over state-of-the-art methods and over several hand-optimized quantum circuits for instances of AES, SHA, and floating-point arithmetic.

1 INTRODUCTION
Logic networks are the central data structure in logic optimization algorithms, which have been widely applied for technology-independent optimization in electronic design automation applications [9, 38, 54]. Roughly speaking, the number of logic gates in a logic network corresponds to the size of a physical implementation, while the number of logic levels corresponds to its delay.

In recent years, the domain of applications for logic optimization has broadened to also target areas such as cryptography [7] and fault-tolerant quantum computing (see, e.g., [28, 29, 50, 52]). Logic networks are typically represented over a gate set consisting of 2-input AND gates, 2-input XOR gates, and inverters, called XOR-AND graphs (XAGs), in which only the AND gates contribute to the cost functions. The multiplicative complexity (MC, [48]) and the multiplicative depth (MD, [11]) of a Boolean function are two important theoretical metrics. The multiplicative complexity is the smallest number of AND gates necessary in any XAG that represents the function. Similarly, the multiplicative depth of a function is the smallest critical path (only considering AND gates) in any XAG that represents the function. We also refer to the length of the critical path (only considering AND gates) as AND-depth.

Multiplicative complexity and depth play important roles in cryptography and fault-tolerant quantum computing. A low multiplicative complexity corresponds to a higher vulnerability to some cryptographic attacks. In fault-tolerant quantum computing, the multiplicative complexity provides an upper bound on the number of expensive quantum operations as well as the number of qubits [31]. Furthermore, the multiplicative depth corresponds to the execution time of a quantum algorithm [32]. Computing the multiplicative complexity of a Boolean function $f$ is expensive. It has been shown that no algorithm exists to compute the multiplicative complexity that is polynomial in the size of the truth table for $f$ [17] if one-way functions [26] exist. We are not aware of any theoretical results concerning the multiplicative depth.

We refer to the number of AND gates and the AND-depth of an XAG by MC/MD of an XAG, respectively. We use just MC or MD if it is clear from the context whether we refer to the MC/MD of a Boolean function or to the MC/MD of a logic network. We note that the latter provides an upper bound to the former.

Thus, many heuristics have been proposed that reduce the MC of an XAG (see, e.g., [7, 14, 43, 55, 56]), aiming to arrive at tighter upper bounds on the MC of the function being implemented. Similarly, some heuristics have been proposed that aim to reduce the multiplicative depth [5, 11]. In this paper, we introduce a logic synthesis algorithm to reduce the MD of logic networks. Our algorithm is based on dynamic programming and makes use of cut enumeration [15], tree balancing [35], as well as ESOP [46] and ESPP [21] representations.

Contributions. We present a fully automatic logic synthesis algorithm that reduces the multiplicative depth of logic networks. We present benchmarks demonstrating that our algorithm is capable of reducing the MD by up to 3x for depth-optimized logic networks and up to 9x for MC-optimized logic networks. As a result, also the quantum circuits derived from our depth-optimized networks feature depths that are significantly smaller than state-of-the-art circuit designs. Crucially, these improvements in depth are possible without increasing the number of qubits significantly.

2 PRELIMINARIES
2.1 Logic networks
In this work, we consider XOR-AND graphs (XAGs), which are logic networks consisting of 2-input AND gates, 2-input XOR gates. Such logic networks can represent all 0-preserving Boolean functions, i.e., functions $f$ for which $f(0, \ldots, 0) = 0$. We are interested in logic networks that minimize the maximum number of AND gates on any path from an input to an output as a primary cost criteria,
and the number of overall AND gates as a secondary cost criteria. Functions $f$, which are not 0-preserving, can be realized by finding an XAG for $\bar{f}$ and then inverting the output. Restricting to have inversions only at the outputs does not affect the AND gates in the circuit, as all inner inversions can be propagated to the outputs by only using XOR gates [48].

Formally, we model an XAG for a single-output Boolean function $f$ over $n$ variables $x_1, \ldots, x_n$ as a sequence of steps, or gates,

$$x_i = f_{j_i} \circ \alpha_i x_{j_i}$$

for $n < i \leq n + r$, and $\alpha_i \in \{0, \lambda\}$. The values $1 \leq j_{1i} < j_{2i} < i$ point to primary inputs or previous steps in the network. The function value is computed by the last step $f = x_{n+r}$. This model is readily extended to multi-output Boolean functions, by associating each output function with some step in the network. The logic level of a primary input or gate $i$ is defined as

$$\ell_i = \begin{cases} 0 & \text{if } i \leq n, \\ \max(\ell_{j_{1i}}, \ell_{j_{2i}}) & \text{if } i > n \text{ and } \alpha_i = \lambda, \\ \max(\ell_{j_{1i}}, \ell_{j_{2i}}) + 1 & \text{if } i > n \text{ and } \alpha_i = \lambda. \end{cases}$$

The depth of an XAG is $d = \max(\ell_i | 1 \leq i \leq n + r)$, the largest level among all gates. In other words, the logic level of a step is the earliest possible time in which a step must be computed, if we aim at parallelizing the evaluation of a logic network. Similarly, we define the reverse logic level $\ell'_i$ as the latest possible time in which step $i$ must be computed while not increasing the depth of the logic network.

### 2.2 Cut enumeration

Many logic optimization algorithms are based on applying local changes to small subnetworks instead of considering the whole logic network at once. An important family of single-rooted subnetworks are cuts. Formally, a cut $C$ of a step $i$ in a logic network is a set of steps, called leaves, such that (i) every path from step $i$ to a primary input visits at least one leaf, and (ii) each leaf is contained in at least one path. Step $i$ is called the root of the cut and each cut represents a subgraph that includes the root $i$ and some internal steps, and has the leaves as primary inputs. A cut is $k$-feasible (referred to as $k$-cut), if $|C| \leq k$, i.e., it has at most $k$ leaves.

Cut enumeration [15] is an algorithm that computes all or a subset of all $k$-cuts for each step in a network. It constructs a mapping $\text{CUTS}(i)$ that maps each step to a set of cuts using the following recursive procedure:

$$\text{CUTS}(i) = \begin{cases} \{\{i\}\} & \text{if } i \leq n, \\ \{\{i\}\} \cup \{C_1 \cup C_2 | C_1 \in \text{CUTS}(j_{1i}), C_2 \in \text{CUTS}(j_{2i}), s.t. |C_1 \cup C_2| \leq k\} & \text{otherwise}. \end{cases}$$

Cuts $\{\{i\}\}$ for root $i$ are called trivial cuts. Note that these are essential, since otherwise the leaves of cuts can only be primary inputs. Cut enumeration can also compute the function $\text{FUNC}(i, C)$ represented by a $C$ for root $i$, by assigning $\text{FUNC}(i, \{\{i\}\}) = x_i$ for all trivial cuts, and

$$\text{FUNC}(i, C) = \text{FUNC}(j_{1i}, C_1) \circ \alpha_i \text{FUNC}(j_{2i}, C_2)$$

if $C$ was constructed using $C_1$ and $C_2$ in (3). Support-normalized truth tables are typically used to represent the cut functions; e.g., truth tables for cut functions $x_1 \land x_3$ and $x_4 \land x_5$ are both represented by the 4-bitstring $1001_2$. To which variables the truth table refers can be determined from the cut’s leaves.

### 2.3 Exclusive sum-of-products

An ESOP for an $n$-variable Boolean function $f(x_1, \ldots, x_n)$ has the form

$$f(x_1, \ldots, x_n) = \bigoplus_{j=1}^{m} (x_1^{p_{j,1}} \land \cdots \land x_n^{p_{j,n}})$$

for some $m$ and polarities $p_{j,i}$, which take values from 0 to 2. Their meaning is that $x_i^{p_{j,i}} = \bar{x}_i$, $x_i^{p_{j,i}} = x_i$, and $x_i^{p_{j,i}} = 1$. We call $x_i^0$ a negative literal, $x_i^1$ a positive literal, and $x_i^{2}$ an empty literal. If $m = 0$, we define $f(x_1, \ldots, x_n) = 0$. The constant-1 function can be represented by an ESOP where $m = 1$ and $p_{1,1} = \cdots = p_{n,1} = 1$.

Each term $\{x_1^{p_{j,1}} \land \cdots \land x_n^{p_{j,n}}\}$ is called a cube of degree $d_j = |\{i | p_{i,j} \neq 2\}|$. It can be regarded as an $(n - d_j)$-dimensional subslice of the $n$-dimensional hypercube, in which the $2^d$ vertices correspond to all bitstrings of length $n$. We require that no cube occurs more than once in an ESOP. The degree of the ESOP is $\max_{1 \leq j \leq m} d_j$.

An ESOP in which $p_{i,j} \neq 0$ for all $1 \leq i \leq n$, $1 \leq j \leq m$ is called the algebraic normal form of $f$. It is unique up to permutation of the cubes. The degree of the algebraic normal form is called the algebraic degree of $f$ and is a lower bound for the degree of any ESOP for $f$. An ESOP can be translated into the algebraic normal form by replacing each cube with $2^d$ cubes in which all $l = |\{i | p_{i,j} = 0\}|$ negative literals are replaced by all combinations of positive and empty literals.

Various exact and heuristic algorithms [8, 16, 20, 37, 40, 44, 46, 47, 53] exist to find ESOPs for Boolean functions, where the primary cost function is the number of cubes in the ESOP and the secondary cost function is the total number of non-empty literals. The positive impact of ESOP expressions to our work is mainly that they have a small depth, thereby having the potential to reduce the multiplicative depth, however, they likely introduce a lot of AND gates to express the cubes. An ESOP optimization algorithm that targets the number of literals as primary cost would therefore be a better fit for our application.

### 2.4 Quantum computing

A quantum computer contains quantum bits, so-called qubits, to which quantum gates are applied in order to solve a computational task. It is controlled by a classical computer running a quantum program, which consists of both classical and quantum instructions: classical instructions are executed by the (classical) host computer, and quantum instructions get sent to the quantum co-processor for execution. In each computational step, the classical computer decides on the sequence of quantum instructions to be executed on the co-processor. Such sequences can be depicted as quantum circuits. The circuit diagram is read from left to right, with each horizontal line representing a qubit, and quantum gates are represented as boxes/symbols on these lines. Fig. 2 shows a quantum circuit that computes the majority-of-5 function and is derived from the logic network in Fig. 1. The circuit consists of CNOT gates $\text{\textdagger}$, $\text{\textdagger}$,
AND gates \[\text{AND}\] as well as uncomputing AND gates \[\text{AND}\]. CNOT gates act on two qubits and compute the XOR of both qubit values onto the lower (target) qubit, leaving the upper (control) qubit unchanged. The AND gate computes a 1 on a newly initialized target qubit, if and only if the two control qubits are 1. The uncomputing AND gate expects that the target qubit is 1 and releases the target qubit in a clean state such that it can be used for subsequent computations.

In this paper, we target quantum computing running a protocol for fault-tolerance, which is necessary to run quantum algorithms with more than a few thousand operations, e.g., for chemistry simulations of practical interest [42]. In this setting, the focus of circuit optimization shifts away from two-qubit gates (e.g., for NISQ devices [41]) toward gates that require distillation. In particular, when the surface code is used, the so-called T-gate incurs a large overhead [2, 39]. In fault-tolerant quantum computing, the cost of CNOTs are typically neglected. The AND gate has a T-count of 4 and a T-depth of 1, if one additional helper qubit is used for its implementation [23] (otherwise, it can be implemented with a T-depth of 2 without the use of a helper qubit). The uncomputing AND gate requires no T-gates.

Previous work [31] focused on reducing the number of costly T-gates. Instead, we aim to shorten the time to solution by reducing the T-depth instead.

3 MULTIPLICATIVE DEPTH REDUCTION

In this section, we introduce various methods that reduce the multiplicative depth of logic networks. Then, we present a procedure to map these networks to quantum circuits while maintaining depth improvements.

3.1 Cut-based balancing

Algorithm 1 describes a generic balancing algorithm based on dynamic programming and cut enumeration inspired by [35]. It takes as input a logic network for an \(n\)-variable Boolean function with \(r\) steps and returns a new depth-optimized logic network. Traversing all steps \(i\) in topological order, it computes depth-optimized candidates for each cut \(C\) of \(i\), and stores the best candidate in a mapping \(\text{BEST}(i)\). The output of the depth-optimized network is \(\text{BEST}(n + r)\) after all steps have been visited. For each cut \(C\) of step \(i\), the algorithm tries to resynthesize the cut function \(\text{FUNC}(i, C)\) with the target to reduce the level of step \(i\). For this purpose, it assumes the best candidates for the cut’s leaves.

The algorithm uses a balance function to resynthesize the cut function. It is therefore generic and can be customized by applying various resynthesis procedures. One possible resynthesis procedure is presented in [35]. It computes a sum-of-products (SOP) representation for the cut function and then translates each term in the SOP into a weight-balanced tree of AND gates, as well as all terms into a weight-balanced tree of OR gates. Our work adapts this method by using an ESOP representation instead, where the outer XOR operations do not contribute to the logic network’s multiplicative depth.

3.2 ESOP balancing

In this section we discuss a rebalancing algorithm based on ESOP forms, which can be used in Algorithm 1. ESOP forms offer a potentially low-depth implementation as an XAG. For the sake of a simpler description of the algorithm, we assume that the ESOP form is given in algebraic normal form, however, in the implementation we consider ESOP forms that also contain negative literals, since they allow for a more compact representation.

Given a \(k\)-cut \(C = \{l_1, \ldots, l_k\}\) of root \(i\) with cut function

\[\text{FUNC}(i, C) = f(\hat{x}_1, \ldots, \hat{x}_k),\]
where \( \hat{x}_i = \text{BEST}(s_i) \) with corresponding level \( \hat{L}_i \). If we are given an ESOP for \( f \) with \( m \) cubes, then each cube is translated into a tree of 2-input AND gates that is balanced with respect to the leaf levels. Then all outputs of these AND-trees are combined by a tree of 2-input XOR gates, which does not add to the multiplicative depth.

### 3.3 ESPP optimization

An exclusive sum-of-pseudoproducts (ESPP [21]) for an \( n \)-variable Boolean function \( f(x_1, \ldots, x_n) \) has the form

\[
f(x_1, \ldots, x_n) = \bigoplus_{j=1}^{m} L_{0, j} \land L_{1, n-1, j}
\]

where \( L_i = b_1x_1 \oplus \cdots \oplus b_nx_n \) when \( i = (b_n \cdots b_1) \) is the linear function (or parity function) that contains variables according to the positions of 1s in the binary expansion of \( i \). The polarity variables \( p_{i,j} \) play the same role as defined for ESOP forms, i.e., the parity function \( L_i \) in term \( j \) is negated if \( p_{i,j} = 0 \), used as is if \( p_{i,j} = 1 \), and omitted if \( p_{i,j} = 2 \). The terms in (6) are called pseudoproducts [27]. Note that each ESPP is an ESOP, but an ESPP is only an ESOP if \( (p_{i,j} \neq 2) \rightarrow (\nu(i) = 1) \) (where \( \nu(i) \) is the sideways sum of \( i \), i.e., the number of 1s in its binary expansion).

The authors presented an exhaustive search algorithm to find small ESPPs in [21], and some theoretical investigations on the form have been conducted [49]. However, to the best of our knowledge no efficient heuristic optimization algorithm for ESPPs has been presented.

We implemented a simple heuristic Greedy minimization algorithm to minimize the number of terms in an ESPP. The algorithm iteratively merges cubes to increase the use of linear functions as cube literals, thereby minimizing the number of AND operations. The algorithm starts with an initial ESPP form that corresponds to an ESOP form, extracted from a cut function. It then checks whether there exists two distinct terms with indices \( j_1 \) and \( j_2 \) such that there exist two indices \( 0 \leq i_1, i_2 < 2^n \) such that \( p_{i_1,j_1} = p_{i_2,j_2} = 2 \) but \( p_{i_1,j_1} \neq 2 \) and \( p_{i_2,j_2} \neq 2 \), and for all other indices \( i \notin \{i_1, i_2\} \), it holds that \( p_{i,j} = p_{i_1,j_1} \). Then, the two terms can be combined into a single term \( i \), with \( p_{i,j} = p_{i_1,j_1} \) for all \( i \notin \{i_1, i_2\} \) and

\[
p_{i_1,j_1} = p_{i_2,j_2} = 2
\]

if \( p_{i_1,j_1} = 1 \). If \( p_{i_1,j_1} = 0 \), then two terms can be found. In our implementation, empty parity functions are not explicitly stored, and therefore this procedure can be efficiently implemented.

### 3.4 Mapping to quantum circuit

Given a logic network over the gate basis \( \{\land, \lor, \neg\} \), it is straightforward to generate a quantum circuit that computes the same function: Each \( \land \) node in the network can be mapped to a Toffoli that writes the output into an extra qubit starting in \([0]\); each \( \lor \) and \( \neg \) node can be computed inplace using a (controlled) NOT gate [31].

While the resulting quantum circuit computes the same function, a significant amount of parallelism is lost due to input-dependencies. As a remedy, we copy the inputs of those gates that can be executed in parallel, thus removing these dependencies [32].

### 4 EXPERIMENTAL RESULTS

We use various arithmetic and random-control functions from [1] as well as cryptographic functions and IEEE floating-point operations [4] as benchmarks for our algorithm. Our algorithm has been implemented in \( \text{C++} \) on top of the EPFL logic synthesis libraries [51]. All experiments were run on a Microsoft Azure virtual machine, on a general purpose Standard D8s v3 size configuration, running on an Intel Xeon Platinum 8171M 2.40GHz CPU with 32 GiB memory and Ubuntu 18.04.

We choose two different baselines as starting points, heavily optimized XAGs for low MC (Min. MC baseline) and heavily optimized AIGs (And-inverter graphs) for low (general) logic network depth. The Min. MC baseline is obtained using the MC optimization algorithm in [56].\(^1\) The Min. MC baseline is obtained by calling the ABC [10] optimization scripts \texttt{resyn2rs} (depth-preserving size optimization [33, 36]), followed by if \( K = 6 \), and \( -y \) (AIG depth optimization [58]), followed by another round of \texttt{resyn2rs}, each run until depth is no longer improved.

### 4.1 Multiplicative-depth optimization

As a first step, we apply ESOP-balancing with a cut size of 6 and exorcism [37] to obtain ESOPs for the cut functions to the chosen benchmarks for both baselines. We call the algorithm repeatedly until no further reduction in the multiplicative depth can be obtained. We report the results in Table 1. For the EPFL benchmarks we list the currently best-known results for multiplicative depth obtained from the state-of-the-art multiplicative depth optimization approach in [5, 11]. That approach has not been applied to the cryptographic and floating-point operations. For each baseline we

\(^1\)The cryptographic and floating-point operations were not further optimized, as they are already optimized for MC.
Lowering the $T$-depth of Quantum Circuits By Reducing the Multiplicative Depth Of Logic Networks

| Benchmark          | State-of-the-art [5, 11] | Min. MC baseline | Min. depth baseline |
|--------------------|---------------------------|------------------|--------------------|
|                    | MC | MD | Run-time | MC (before) | MD (before) | Run-time | MC (before) | MD (before) | Run-time |
| Arithmetic functions [1] |               |          |          |           |            |       |           |            |       |
| adder              | 16378 | 10 | 125.00 | 481 (128) | 34 (128) | 0.15 | 2761 (1742) | 12 (14) | 10.13 |
| bar                | 4193 | 0.70 | 1303 (832) | 4 (7) | 0.33 | 3516 (3334) | 8 (11) | 2.42 |
| div                | 190855 | 532 | 3731.00 | 150765 (56635) | 973 (2243) | 26.18 | 120327 (120327) | 523 (620) | 541.33 |
| hyp                | 135433 | 15230 | 172000.00 | 120765 (56635) | 4428 (8784) | 166.07 | 780220 (145767) | 1287 (1558) | 324.31 |
| log2               | 31573 | 129 | 94.00 | 34133 (10906) | 104 (201) | 778.78 | 83177 (33951) | 114 (171) | 130.33 |
| max                | 7666 | 26 | 14.50 | 3839 (890) | 93 (252) | 1.81 | 8368 (4027) | 25 (28) | 4.17 |
| multiplier         | 23059 | 532 | 3731.00 | 15138 (7653) | 65 (149) | 13.50 | 39628 (28331) | 56 (86) | 77.72 |
| Random control [1] |                   |          |          |           |            |       |           |            |       |
| arbiter            | 5183 | 10 | 43.00 | 3128 (1174) | 13 (50) | 2.10 | 7276 (6205) | 11 (12) | 1.35 |
| cavlc              | 667 | 0.00 | 447 (394) | 7 (11) | 1.15 | 564 (576) | 8 (10) | 0.45 |
| ctrl               | 109 | 5 | 0.10 | 54 (45) | 4 (5) | 0.10 | 77 (80) | 4 (8) | 0.06 |
| dec                | 304 | 3 | 0.00 | 328 (328) | 3 (3) | 0.08 | 292 (292) | 3 (3) | 0.02 |
| i2c                | 1213 | 7 | 0.10 | 816 (557) | 7 (11) | 0.87 | 1122 (1007) | 7 (8) | 0.37 |
| int2float          | 216 | 7 | 0.00 | 104 (85) | 6 (11) | 0.87 | 184 (190) | 7 (8) | 0.13 |
| mem_ctrl           | 54816 | 40 | 85.00 | 9983 (4695) | 14 (39) | 17.56 | 78044 (37519) | 35 (41) | 20.37 |
| priority           | 876 | 102 | 0.50 | 442 (323) | 11 (95) | 1.08 | 522 (479) | 10 (13) | 0.28 |
| router             | 198 | 11 | 0.00 | 116 (93) | 8 (13) | 0.10 | 227 (196) | 10 (12) | 0.19 |
| voter              | 4288 | 30 | 112.42 | 7335 (4257) | 26 (40) | 31.95 | 3255 (6716) | 17 (48) | 6.14 |
| Cryptographic functions [4] |               |          |          |           |            |       |           |            |       |
| AES-128            | 8400 (6400) | 50 (60) | 5.94 | 33953 (85547) | 80 (299) | 65.52 |
| AES-192            | 9408 (7168) | 60 (72) | 5.98 | 39533 (96079) | 99 (359) | 55.39 |
| AES-256            | 11592 (8832) | 70 (84) | 7.49 | 53775 (120627) | 123 (417) | 90.26 |
| Keccak-f           | 38400 (38400) | 24 (24) | — | 38630 (567395) | 28 (266) | 129.00 |
| SHA-256            | 22573 (22573) | 1607 (1607) | — | 450447 (296951) | 1519 (1936) | 247.16 |
| SHA-512            | 57947 (57947) | 3303 (3303) | — | 1988586 (831166) | 2383 (2894) | 1489.64 |
| IEEE floating-point operations [4] |               |          |          |           |            |       |           |            |       |
| FP-add             | 16721 (5384) | 96 (235) | 9.93 | 27541 (15879) | 64 (83) | 15.42 |
| FP-div             | 3829444 (82265) | 1646 (3619) | 2994.35 | 732932 (200112) | 885 (1157) | 400.12 |
| FP-eq              | 315 (315) | 9 (9) | — | 220 (336) | 9 (10) | 0.02 |
| FP-f2i             | 3290 (1467) | 24 (94) | 3.01 | 3405 (2881) | 21 (29) | 3.41 |
| FP-mul             | 23886 (19614) | 92 (129) | 14.78 | 62254 (47213) | 87 (140) | 54.51 |
| FP-sqrt            | 494657 (91499) | 3763 (6307) | 2981.18 | 893849 (264130) | 1877 (2374) | 506.28 |

Our algorithm can improve the best-known results in 18 out of 20 cases. For the arithmetic functions, the largest MD reductions were obtained when applying our approach to the Min. depth baseline, whereas for the random control functions, the Min. MC baseline turns out to be the better starting point. Note that in some cases (e.g., hyp and priority) we obtain a $10\times$ improvement over the state of the art. For the cryptographic and floating-point functions, we can improve the MD compared to both baselines for all benchmarks except for Keccak-f. Because we use heavily-optimized networks as the baseline, we do not expect large gains for cryptographic functions, especially since MC and MD are important quantities in cryptography. In contrast, we find depth-reductions of up to $3\times$ for floating-point operations (e.g., FP-add and FP-f2i) with only moderate increases in MC.

### 4.2 T-depth optimization

In a second step, we map our depth-optimized XAGs to quantum circuits using two straightforward heuristics for upper-bounding the number of qubits: the as soon as possible (ASAP) heuristic computes all AND gates in parallel that have the same logic level and the as late as possible (ALAP) heuristic computes all AND gates in parallel that have the same reverse logic level. We present the resulting $T$-counts, $T$-depths, and qubit estimates in Table 2. For each cryptographic function and floating-point operation, we report the two quantum circuits with the fewest number of qubits (first row) and the lowest $T$-depth (second row). The corresponding
Figure 3: These plots contain various resource estimates that can be found in the literature, together with all Pareto-optimal results for our approach that we obtained during the experimental evaluation. These include also results from intermediate optimization steps.

XAG and heuristic (ASAP or ALAP) is given in the last column. Compared to state-of-the-art, manually-crafted quantum circuit designs, we achieve significant reductions in depth without dramatically increasing the qubit requirements. A comparison of our automatically-generated designs to a variety of state-of-the-art circuits for several cryptographic and floating-point functions is given in Fig. 3 with best T-depth state-of-the-art implementations explicitly reported in Table 3.

We note that, in addition to reduced circuit depths compared to the state of the art, our approach has the clear advantage that it is completely automatic. This stands in stark contrast to the circuits found in the literature, since those are manual designs that were not created by the push of a button.

5 CONCLUSIONS

In this work we presented dynamic programming algorithm to minimize the multiplicative depth of XAGs that makes use of cut enumeration, tree balancing, as well as ESOP and ESPP representations. We can report significant improvement to the state-of-the-art MD optimization algorithms in [5, 11]. We used our algorithm to
Table 2: Estimates for $T$-depth optimized quantum circuits obtained from depth-optimized XAGs. We report quantum circuits that achieve the smallest number of qubits (first row) and the lowest $T$-depth (second row) over all $T$-depth optimized circuits.

| Benchmark | T-count | T-depth | Qubits | Instance |
|-----------|---------|---------|--------|----------|
| AES-128   | 25600   | 60      | 7324   | Min. MC baseline (ASAP) |
| AES-128   | 33600   | 50      | 9384   | Min. MC opt (ASAP) |
| AES-192   | 28672   | 72      | 8156   | Min. MC baseline (ASAP) |
| AES-192   | 37632   | 60      | 10456  | Min. MC opt (ASAP) |
| AES-256   | 35328   | 84      | 9884   | Min. MC baseline (ASAP) |
| AES-256   | 46368   | 70      | 12704  | Min. MC opt (ASAP) |
| Keccak-f  | 153600  | 24      | 46400  | Min. MC baseline (ASAP) |
| SHA-256   | 90292   | 1607    | 23684  | Min. MC baseline (ASAP) |
| SHA-256   | 1801788 | 1519    | 458974 | Min. depth opt (ASAP) |
| SHA-512   | 231788  | 3303    | 60448  | Min. MC baseline (ASAP) |
| SHA-512   | 7954344 | 2383    | 2008595| Min. depth opt (ASAP) |

IEEE floating-point operations

| Benchmark | T-count | T-depth | Qubits | Instance |
|-----------|---------|---------|--------|----------|
| FP-add    | 21384   | 235     | 5969   | Min. MC baseline (ALAP) |
| FP-add    | 100832  | 64      | 28154  | Min. depth opt (ASAP) |
| FP-div    | 290848  | 3604    | 81066  | Min. MC baseline (ALAP) |
| FP-div    | 3054524 | 885     | 79288  | Min. depth opt (ALAP) |
| FP-eq     | 880     | 9       | 655    | Min. depth opt (ALAP) |
| FP-eq     | 1260    | 9       | 976    | Min. MC baseline (ALAP) |
| FP-f2i    | 5832    | 96      | 1821   | Min. MC baseline (ALAP) |
| FP-f2i    | 13620   | 21      | 4846   | Min. depth opt (ALAP) |
| FP-mul    | 76368   | 118     | 26890  | Min. MC baseline (ALAP) |
| FP-mul    | 249052  | 87      | 69347  | Min. MC baseline (ALAP) |
| FP-sqrt   | 315924  | 6498    | 84017  | Min. MC baseline (ALAP) |
| FP-sqrt   | 3575396 | 1877    | 901087 | Min. depth opt (ALAP) |

Table 3: Estimates from related work

| Benchmark | T-count | T-depth | Qubits | Comment |
|-----------|---------|---------|--------|---------|
| AES-128   | 1006064 | 50688   | 984*   | 1       |
| AES-192   | 1204224 | 44352   | 1112*  | 1       |
| AES-256   | 1505280 | 59904   | 1336*  | 1       |
| AES-256   | 118580  | 7520    | 864*   | 2       |
| AES-192   | 137060  | 6560    | 896*   | 2       |
| AES-256   | 166320  | 8640    | 1232*  | 2       |
| AES-128   | 54400   | 120     | 1785*  | 2       |
| AES-192   | 60928   | 120     | 2105*  | 2       |
| AES-256   | 75072   | 126     | 2425*  | 2       |
| Keccak-f  | 24640   | 33      | 3200*  | 3       |
| SHA-256   | 30336   | 938*    | 1      |
| SHA-256   | 228992  | 70400   | 2402*  | 3       |
| FP-add    | 26348   | 7224    | 268*   | 3       |
| FP-mul    | 122752  | 52116   | 315*   | 3       |

A * indicates that this value is better compared to the best value reported in Table 2.

1 Authors report no Toffoli-count or $T$-count; $T$-depth is derived from reported Toffoli-depth by multiplication with 3 [2]; authors report six different candidates, from which we picked the one with the best $T$-depth.

2 $T$-count and $T$-depth are derived from reported Toffoli-count and Toffoli-depth in the paper.

3 The floating-point designs in the paper are not IEEE-compliant and do not account for special cases or denormalized numbers.

REFERENCES

[1] Luca Caetano Amarí, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL combinational benchmark suite. In Int’l Workshop on Logic and Synthesis.

[2] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. 2013. A Meet-in-the-Middle Algorithm for Fast Synthesis of Depth-Optimal Quantum Circuits. IEEE Trans. on CAD of Integrated Circuits and Systems 32, 6 (2013), 818–830. https://doi.org/10.1109/TCAD.2013.2244643

[3] Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John M. Schanck. 2016. Estimating the Cost of Generic Quantum Pre-image Attacks on SHA-2 and SHA-3. In Int’l Conf. on Selected Areas in Cryptography. 317–337. https://doi.org/10.1007/978-3-319-69453-5_18

[4] David Archer, Victor Arribas Abril, Pieter Maene, Nele Mertens, Danilo Sijacic, and Nigel Smart. [n.d.]. 'Bristol Fashion' MPC circuits. https://homes.esat.kuleuven.be/~nsmart/MPC/ https://homes.esat.kuleuven.be/~nsmart/MPC/

[5] Pascal Aubry, Sergiu Carpov, and Renaud Sirdey. 2020. Faster Homomorphic Encryption is not Enough: Improved Heuristic for Multiplicative Depth Minimization of Boolean Circuits. In The Cryptographers’ Track at the RSA Conference. 345–363. https://doi.org/10.1007/978-3-030-40186-3_15

[6] C. Leonard Berman, David J. Hathaway, Andrea S. LaPaugh, and Louise H. Trevillyan. 1990. Efficient techniques for timing correction. In Int’l Symp. on Circuits and Systems. 415–419. https://doi.org/10.1109/ISCAS.1990.112064

[7] Jean Boyar, Philip Matthews, and René Peralta. 2013. Logic Minimization Techniques with Applications to Cryptology. Journal of Cryptology 26, 2 (2013), 269–312. https://doi.org/10.1007/s00145-012-9124-7

[8] David Brand and Tustum Sasso. 1993. Minimization of AND-EXOR Expressions Using Rewrite Rules. IEEE Trans. on Computers 42, 5 (1993), 568–576. https://doi.org/10.1109/12.223676

[9] Robert K. Brayton, Gary D. Hachtel, and Alberto L. Sangiovanni-Vincentelli. 1990. Multilevel logic synthesis. Proc. IEEE 78, 2 (1990), 264–300.

not increase. This allows to reduce the $T$-count and qubit count in corresponding quantum circuits without increasing the $T$-depth.
