Optimizing Quantum Circuit Synthesis with Dominator Analysis

Giacomo Lancellotti
Politecnico di Milano-DEIB
Milano, Italy
giacomo.lancellotti@polimi.it

Giovanni Agosta
Politecnico di Milano-DEIB
Milano, Italy
giovanni.agosta@polimi.it

Alessandro Barenghi
Politecnico di Milano-DEIB
Milano, Italy
alessandro.barenghi@polimi.it

Gerardo Pelosi
Politecnico di Milano-DEIB
Milano, Italy
gerardo.pelosi@polimi.it

***Abstract*—Quantum circuit synthesis translates a classical Boolean function into an equivalent quantum circuit. The synthesis is solvable by playing the *reversible pebble game* on the logic network of the function. However, optimal solutions are impractical for large networks, affecting the number of qubits and gates of the final quantum circuit. In this work, we improve the solution of the reversible pebble game by leveraging dominance relations on the directed acyclic graph of the classical function, reducing qubits needed for syntheses. The proposed algorithm exposes a tunable tradeoff between the available number of qubits and the circuit size expressed as T-count and T-depth. We experimentally validate our methodology on cryptographic and arithmetic benchmarks, reporting reductions between 37% and 83% in qubit number with respect to Bennet syntheses algorithms and complete the majority of our syntheses in less than a second, improving on the current running times of state of the art approaches.**

***Index Terms*—Logic and circuit design, reversible synthesis, quantum circuits optimization, reversible pebble game**

I. Introduction

Quantum computing has emerged as a key technology due to its promise to address otherwise computationally intractable problems [1]. Despite significant efforts to increase the number of qubits available for computation, it remains a critical bottleneck for running quantum algorithms on problems of practical size. Among the known quantum algorithms [2], many of them, require the implementation of some combinatorial logic, such as complex arithmetic functions, which can demand a large amount of resources. Therefore, synthesizing quantum algorithms using the minimum number of qubits is paramount for enhancing the applicability of quantum computers. In this work, we focus on the problem of automatically synthesizing a multi-level classical logic function into a reversible quantum circuit with a constrained number of qubits. We start from the gate-level view of the logic network in input, which can be easily derived from existing EDA tools, and translate it into an equivalent XOR-AND graph (XAG) format. In his seminal work [3], Charles H. Bennett demonstrated that any irreversible computation can be made reversible and thus synthesized in a quantum circuit, with a constant space overhead, and that the syntheses process is as complex as solving the *reversible pebble game*. The game consists in placing "pebbles" or "markers" on the rooted directed acyclic graph (DAG) of the function to synthesize. In [4], the authors proved that any n-vertex planar DAG, with maximum number k of incoming arcs in any node, can be pebbled using $\mathcal{O}(\sqrt{n}+k\log_2(n))$ pebbles. However, finding the optimal strategy to solve the reversible pebble game on a DAG, i.e., pebbling the root node with the minimum number of pebbles, it is shown to be a challenging and computationally demanding task belonging to the PSPACE-Complete computational class [5].

Contribution. In this work, we introduce a novel strategy to solve the reversible pebble game with a minimal number of pebbles, when applied to the synthesis of a quantum circuit realizing a given classic combinatorial function (specified via a XOR-AND DAG). Our proposal employs the notion of *graph-dominators* from compiler construction theory, to effectively improve on reversible pebble game solvers currently used in the synthesis of quantum circuits. We test the proposed strategy on a set of cryptographic and arithmetic circuits reporting an average saving of 62.6% in qubit number compared to Bennet strategy and comparable results with respect to state-of-the-art pebble game solvers reducing the computational running time.

Related Work. A survey on reversible circuit synthesis is given in [6], [7]. In [8], a SAT solver is used to address the reversible pebble game, it results in a qubit saving of the 52.77% compared to Bennet strategy which is lower compared to ours. In [9], the authors design a heuristic algorithm to tackle the pebble game, in conjunction with pre- and post-processing strategies, which may benefit from an improved pebble game solver. Finally, [10] employs XOR-AND-Inverter graphs and SAT-based heuristics to reduce qubits during the syntheses, though their results are limited to four benchmarks due to the solver's long execution times (up to 1.5 hours per run).

II. Background

A. Quantum Circuit Synthesis

Every single-output Boolean function $f{:}\{0,1\}^n{\to}\{0,1\}$, defined on $n{\geq}1$ binary variables $x_0, x_1, \ldots, x_n$, can be represented as a multiple rooted DAG, where each node is labeled with a logic operation, and arcs denote data dependencies. In this work, we express our function using the functionally complete AND-XOR set, as the equivalent quantum gates for AND and XOR are easily implementable with the universal *Clifford+T* quantum gate set [11]. Using the AND-XOR representation, each node of the DAG has at most two incoming

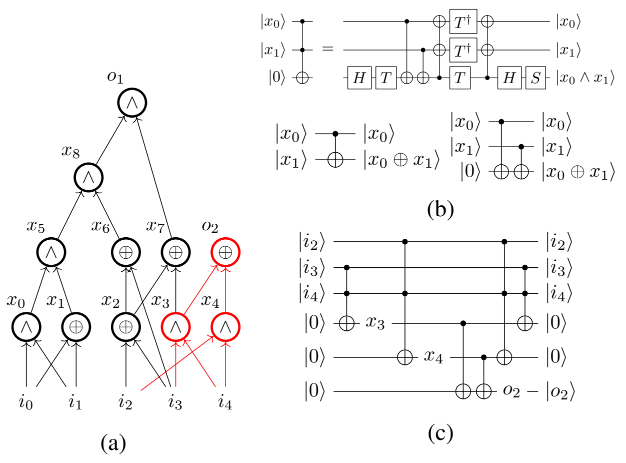

Fig. 1: XOR-AND graph of a combinatorial function, with the cone of influence $\mathcal{C}_{o_2}$ highlighted in red (a). CCNOT with minimal T-depth decomposition and CNOT circuits, w/o and w/ a qubit ancilla (b). Bennet's algorithm result for the syntheses of $\mathcal{C}_{o_2}$(c).

arcs from *immediate predecessors*, and a bounded number of outgoing arcs to *immediate successors*. Each single rooted subDAG represents a *cone of influence* $\mathcal{C}_{o_i}$ of the inputs and intermediate values that contribute to the output at the root node o_i. For instance, in the graph shown in Fig. 1a, the cone of influence for o_2 is $\mathcal{C}_{o_2}=\{o_2, x_3, x_4, i_2, i_3, i_4\}$. The output is computed by following the operations in the DAG, respecting the data dependencies. Quantum circuit synthesis consists in finding a unitary operator U_f, acting on the Hilbert space of $n+1$ qubits, that realizes a reversible version of f [11]. Using quantum circuit formalism, each qubit is represented as a horizontal wire, associated with a quantum state $|x_i\rangle$, where x_i is a binary variable storing a classical bit and computation is performed by applying quantum gates, which modify qubits state. To implement our circuits, we use the universal Clifford+T set, which includes the H, S, CNOT, and T gates. With this set the reversible AND ($\wedge$) and XOR ($\oplus$) gates are realized with then CCNOT and CNOT quantum gates respectively. The circuit representations of the CNOT with and without ancillae qubits and CCNOT with its Clifford+T decomposition are shown in Fig.1b. Since the T-gate is the most costly to implement on quantum hardware [12], the number of T-gates (T-count) and the longest sequential number of T-gates (T-depth) are common figures of merit for quantum algorithms.

B. Pebble Game

The pebble game, is a mathematical abstraction to model resource management for both reversible and irreversible computations. The game involves placing "pebbles" on the nodes of a rooted DAG that represent the operations that must be executed to obtain the chosen output value, following the dependencies represented by the arcs. Each pebble represents a computational resource, such as a memory cell or a qubit, and *pebbling* a node denotes reserving a resource to store the result computed by that node. Conversely, *unpebbling* a node frees

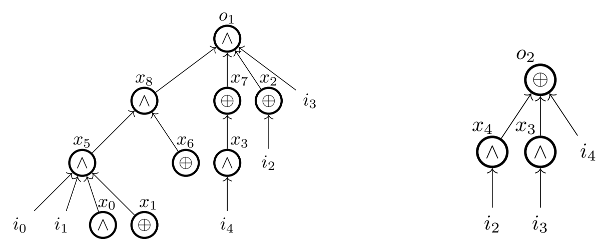

Fig. 2: Post-dominator tree of $\mathcal{C}_{o_1}$ (left) and $\mathcal{C}_{o_2}$ (right), both in the XOR-AND graph shown in Fig.1a.

the associated resources. In the reversible version of the game, introduced by Bennett [3], a node can be pebbled or unpebbled if all of its immediate predecessors are pebbled, ensuring that the computation can be reversed at any time. Given a DAG of a classical function, several pebbling strategies can be used to synthesize the equivalent quantum circuit. One example is Bennett's algorithm [3], which pebbles the nodes in a breadth-first, bottom-up manner. Starting from the inputs, the quantum circuit is built by placing the quantum gates corresponding to each visited node on the assigned qubit. Once the root node is pebbled, the uncomputation of intermediates nodes is done using a breadth-first, top-down strategy. It results in a circuit with the minimum number of gates and depth, however there is no limitation on the number of qubits. In Fig.1c it is shown the quantum circuit for computing $\mathcal{C}_{o_2}$ following Bennett's algorithm. The maximum number of pebbles on the graph at any time is called the *pebble number*, which reflects the maximum resource consumption of the process. Finding the minimum pebble number required to compute a target node on a DAG has been shown to be PSPACE-Complete [5]. As a result, no polynomial-time strategies exist to solve it optimally, except on particular graph topology such as trees [13], and so efficient heuristic based algorithms to minimize the pebble number are of considerable interest.

C. Graph Dominators

The concept of graph dominators, initially used in compiler construction, have found several applications in classical EDA, such as in logic synthesis and verification, or in network delay optimization [14]. In this work, we focus on the notion of *post-dominator*, given a DAG $G=(N, A, i, o)$, with i and o being the initial and output node respectively, a vertex v is defined as a post-dominator for a vertex w if every path from w to o necessarily passes through v. The vertex v is said to be *immediate post-dominator*, if it is the closest vertex among all post-dominators of w. Every node, except the root, has a unique immediate post-dominator. A set of immediate post-dominators forms a directed tree $T(G)$, a.k.a. *post-dominator tree*, rooted at one of the output nodes of the DAG. Figure 2 shows the post-dominator trees of C_{o_1} and C_{o_2}. Intuitively, an immediate post-dominator identifies the point where multiple execution paths converge in the original graph. For example, in Fig.2, node x_5 is an immediate post-dominator for the nodes i_0, i_1, x_0, x_1, also denoted as $\text{dom}(x_5)=\{i_0, i_1, x_0, x_1\}$. An algorithm to compute the dominance relations in linear time as a function N was first introduced in [15].

III. Dominator based synthesis

Algorithm 1: Dominator-based Pebble Game

```
   // Compute the Pebble Game Strategy
1  Procedure PebbleGame
       Input: $\mathcal{C}(N, A)$: single-output DAG, i.e., a cone of influence described by a set of nodes $N$ and a
              set of arcs $A$
              maxR: maximum number of recursive steps
              $\mathtt{n_q}$: maximum number of qubits
       Output: $S$: ordered sequence of $\langle n', a' \rangle$, where $n'$ is a node and $a'$ is an action, denoted in
               turn by a Boolean value as 1="pebble", 0="un-pebble"
       Data: stack: last-in-first-out array, each cell stores a copy of ( $S$, $L$ ), where $L$ is the set of
             actions already attempted for the currently evaluated $S$
2      $o \leftarrow outnode(\mathcal{C})$
3      $Q \leftarrow leaves(\mathcal{C})$ // currently pebbled nodes
4      $S, L \leftarrow \emptyset, \emptyset$
5      $push$(stack, ( $S$, $L$ )
6      while ( $top$(stack) $\neq \bot$ ) do
7          if ( $o \in Q$ ) then return $S$
8          $M \leftarrow$ GetScoredActions($\mathtt{n_q}$, $Q$, $S$, $o$)
9          if ( |stack| = maxR $\vee$ $M = \emptyset$ $\vee$ $M = L$) then
10             $pop$(stack)
11             $S', L' \leftarrow pop$(stack)
12             $S, L \leftarrow S', L' \cup \{last(S)\}$
13         else
14             $\langle n, a, s \rangle \overset{\$}{\leftarrow} \{\langle n, a, s \rangle \mid \forall \langle n', a', s' \rangle \in M, s \geq s' \}$
15             if ( $a = 1$ ) then $Q \leftarrow Q \cup \{n\}$
16             else $Q \leftarrow Q \setminus \{n\}$
17             $S, L \leftarrow S.\langle n, a, s \rangle, \emptyset$
18         $push$(stack, ( $S$, $L$ ))
19     return $\emptyset$ // the algorithm fails
   // Select the best move to execute
20 Procedure GetScoredActions
       Input: $\mathtt{n_q}$: maximum number of qubits
              $Q$: set of nodes linked to assigned qubits
              $S$: ordered sequence of $\langle n', a' \rangle$, where $n'$ is a node and $a'$ is an action, denoted in turn
              by a Boolean value as 1="pebble", 0="un-pebble".
              $o$: output node
       Output: $M$: list of scored actions, where each of them is denoted as $\langle n, a, s \rangle$, $n$: node; $a$:
               action, denoted by a Boolean value as 1="pebble", 0="un-pebble"; $s$: score
21     $M \leftarrow \emptyset$
22     foreach $n \in Q$ do
23         if ( $pred(n) \subseteq Q$ ) then
24             $s \leftarrow$ ScoreAction($\langle n, 0 \rangle$, $Q$, $S$, $o$)
25             $M \leftarrow M \cup \{ \langle n, 0, s \rangle \}$
26         if ( $|Q| < \mathtt{n_q}$ ) then
27             foreach $v \in succ(n)$ do
28                 if ( $pred(v) \subseteq Q$ ) then
29                     $s \leftarrow$ ScoreAction($\langle v, 1 \rangle$, $Q$, $S$, $o$)
30                     $M \leftarrow M \cup \{ \langle v, 1, s \rangle \}$
31     return $M$ // $\emptyset$ if no possible actions
   // Score the move
32 Procedure ScoreAction
       Input: $\langle n, a \rangle$: node and action to be scored;
              $Q$: set of nodes linked to assigned qubits;
              $S$: ordered sequence of $\langle n', a' \rangle$, where $n'$ is a node and $a'$ is an action, denoted in turn
              by a Boolean value as 1="pebble", 0="un-pebble".
              $o$: output node
       Output: $s$: score
33     $s \leftarrow -|\{\langle n', a' \rangle \in S \text{ s.t. } n' = n\}|$
34     if ($a = 1$) then
35         $s \leftarrow s - dist(n, o)$
36         if ($dom(\mathtt{n}) \neq \emptyset$ ) then $s \leftarrow s + |dom(n)|$
37     else
38         $s \leftarrow s + dist(n, o)$
39         if ( $\exists\, v \in Q \mid n \in dom(v)$ ) then $s \leftarrow s + 1$
40     return $s$
```

The proposed strategy takes as input a combinatorial function (expressed as a XAG) and the number of available qubits $\mathtt{n_q}$. As preprocessing step, the input graph is decomposed into smaller single-output cones of influence, $\mathcal{C}_i(N_i, A_i)$ (see Sec.II), by executing a depth-first visit of the main graph in $\mathcal{O}(n_{out}(N_i{+}A_i))$, where n_{out} is the number of original outputs nodes. We first solve the pebble game for the cone with the maximum number of nodes, which accelerates solutions for the others due to fewer constraints from non-overlapping paths in other subDAGs [9]. Cone ordering is done in $\mathcal{O}(n_{out}{+}(max(N_i){-}min(N_i)))$. Dominance relations are computed using the Lengauer-Tarjan algorithm [16], with an asymptotic complexity of $\mathcal{O}(A_i\,\alpha(N_i, A_i))$, where $\alpha(N_i, A_i){\approx}1$ for all N_i, A_i values of practical interest [17]. The sets of predecessors, successors, or post-dominated nodes for a node n, are denoted as $pred(n)$, $succ(n)$, and $dom(n)$.

Algorithm 1 reports the pseudocode of our reversible pebble game solver. We use a backtracking approach where maxR is the maximum number of recursive steps in our pebbling strategy S. S is an ordered sequence of pairs $\langle n, a \rangle$, where n is a node identifier, and a is the action on the node (a=1 for pebbling; a=0 for unpebbling). At line 3, we initialize a set Q to track of pebbled nodes, assuming its maximum size to be less or equal to $\mathtt{n_q}$. A list L stores moves explored in the current solver state given by S, L, saved in a lifo stack. Before backtracking to a different strategy, we first exhaust all possible valid moves not yet in L. In the main loop lines 6-18 we check if the output node o is pebbled. If it is, we return the current strategy (line 7); otherwise, we compute the list of admissible moves M (line 8). Next, we verify if M is non-empty and not fully explored (i.e., $M{\neq}L$). If so, and the maximum stack depth hasn't been reached, we select the move with the highest score (or randomly among the top-scoring moves) (line 14), apply its effects to Q (lines 15-16), and append the selected action to the strategy S. If any condition in line 17 is met, we pop the top of the stack, add the chosen action to the list of attempted moves L', and prepare to push the pair composed of the strategy S' (which is S without its last move) and L' back onto the stack. Finally, the pair S,L is pushed onto the stack before the next iteration (line 18). If the stack is emptied through a sequence of iterations, no admissible solution is found within the given qubit number and stack height constraints, causing the algorithm to terminate (line 19) and return an empty strategy.

The main procedure relies on GetScoredActions to compute the list of admissible moves M, given the state of the exploration (i.e., S, Q, o, $\mathtt{n_q}$). Each element of Q can be pebbled or unpebbled if the results of $pred(n)$ are currently allocated in some qubits (lines 23–30). GetScoredActions has a worst-case time complexity of $\mathcal{O}(|Q| \times |succ(n)|)$. The ScoreAction procedure computes a score s for a move $\langle n, a \rangle$. Since the main procedure selects the move with the maximum (positive) score at each iteration, we design our scoring mechanism to penalize excessive backtracking. To this end, s is initialized to a negative penalty equal to the number of times a node appears in the current strategy S (line 33). For pebbling moves (lines 35–36), we direct the computation towards the output node, applying an additional penalty based on the normalized distance $dist(n, o)$. The heuristic prioritizes pebbling nodes with a large dominated set $dom(n)$; such nodes, once computed, enable the unpebbling of all nodes they dominate. For unpebbling moves, we reverse the distance penalty and add a positive factor if the node to unpebble has a pebbled dominator (lines 38–39). ScoreAction is computed in $\mathcal{O}(1)$.

The overall complexity of PebbleGame is dominated by the cost of GetScoredActions. The total complexity is $\mathcal{O}(\mathtt{maxR} \times \mathtt{n_q} \times |N|)$. Although finding a solution within the maximum exploration depth is not guaranteed, the parameter maxR can be tuned based on the characteristics of the graph. The construction of the quantum circuit involves inserting, for every pebbling/unpebbling move, the corresponding quantum

TABLE I: Synthesis results obtained with the minimum number of qubits. It is also reported the number of qubits required by Bennett [3] strategy, denoted as Q_B. The column $-\Delta\%$ shows the percentage reduction in qubits achieved by our strategy compared to the latter.

Benchmark	**I**	**O**	$\wedge$	$\oplus$	**Q**	**T-c**	**T-d**	**Q_B**	$-\Delta\%$	$t(s)$
Figure 1a	5	2	6	5	10	36	13	16	37.5	0.00
`mx6x31`	12	6	27	30	20	208	50	69	71.0	0.04
`mx7x41`	14	7	40	44	23	300	61	98	76.5	0.06
`mx7x31`	14	7	40	45	23	364	75	99	76.8	0.09
`x8x4x31`	16	8	48	69	37	864	134	133	72.1	0.22
`bm-10`	20	19	52	102	51	928	132	174	70.7	0.52
`bm-30`	60	59	351	687	211	9872	780	1098	80.7	29.50
`bm-40`	80	79	624	1079	287	16572	751	1783	83.9	85.91
`32-lt`	64	1	150	0	139	900	77	214	35.0	0.47
`32-lteq`	64	1	150	0	137	908	86	214	36.0	0.47
`32-add`	64	33	127	61	127	10656	939	252	49.6	6.06
`64-add`	128	65	265	115	255	40608	2384	508	49.8	57.72
`FP-eq`	128	64	315	65	258	2000	220	508	49.2	3.93

gate for the node: CCNOT for $\wedge$ and CNOT for $\oplus$, correctly applied to the reserved qubit. Once the output is pebbled, intermediate results are uncomputed by reversing the strategy. If each cone admits a valid strategy, we merge them into a unique quantum circuit.

IV. Experimental evaluation

We synthesize circuits for multiplication in $GF(2^6)$, $GF(2^7)$, and $GF(2^8)$, using the irreducible polynomials x^6+x^3+1 (`mx6x31`), x^7+x^4+1 (`mx7x41`), and x^7+x^3+1 (`mx7x31`) for $GF(2^6)$ and $GF(2^7)$, and the AES polynomial $x^8+x^4+x^3+x+1$ (`x8x4x31`) for $GF(2^8)$. Additional benchmarks include binary polynomial multiplications (`bm-n`), 32 and 64-bit integer adders (`32-add`, `64-add`), 32-bit integer comparators (`32-lt`, `32-lteq`), and IEEE floating-point comparisons (`FP-eq`). Quantum circuits are synthesized using the CCNOT gate, with the decomposition shown in Fig.1b which has a T-count and T-depth of 4 and 2 respectively, for CNOT we use the implementation with one ancilla. The CCNOT gate uncomputation is realized with the measurement-based approach shown in [18]. We implemented our code in Python 3.12 and tested it on a laptop with 8 GiB of RAM and an Intel Core i5-8259U CPU clocked at 2.3 GHz running macOS 14.4. For our benchmarking, we set the initial number of qubits equal to the number of nodes in the largest cone of the circuit, setting the maximum exploration depth to $\mathtt{maxR}{=}10^5$. We report in Tab.I the synthesis results achieved with the lowest number of qubits, additionally, we report the qubits utilized by Bennett's strategy Qubit_B, along with the savings in percentage difference of our approach $-\Delta\%$. On average, we observe a 62.6% qubit saving for the circuits under consideration. In comparison to other works, the authors of [9] solve the pebble game on a DAG with multi-input quantum gates, which are decomposed in a second pass. This makes the impact on qubit usage unclear. In [8], the authors report an average qubit reduction of 52.77% compared to Bennett, which is lower than our result. While their approach uses a SAT solver with a complexity $\mathcal{O}(|N|^2)$, we achieve similar complexity by tuning the parameters `maxR` and $\mathtt{n_q}$. The work in [10] addresses the pebble game on XAG by reducing the multiplicative complexity (AND gates) to minimize T-gate usage. Although their qubit usage is similar to ours, their method requires approximately 1.5 hours per circuit, while our approach processes most benchmarks in sub-seconds and completes all within two minutes (see Tab.I). Our higher T-count arises from processing each cone separately, which optimizes execution speed but increases T-count and T-depth due to redundant computation and uncomputation of shared nodes across cones.

V. Concluding remarks

In this work we address the problem of automatically synthesizing combinatorial Boolean functions as reversible quantum circuits, under the constraint of a limited number of available qubits. We introduce a novel dominator-based heuristic for efficiently solving the reversible pebble game. Our algorithm achieves an average reduction of 62.6% in qubit usage compared to Bennett's strategy, while obtaining comparable results with existing methods and improving running time by over two orders of magnitude.

Acknowledgement

The activity was partially funded by the *Italian Centro Nazionale di Ricerca in HPC, Big Data e Quantum computing - SPOKE 10*

References

[1] IBM Institute for Business Value, "The Quantum Decade," 2023.
[2] A. Montanaro, "Quantum algorithms: an overview," *npj Quantum Inf.*, 2016.
[3] C. H. Bennett, "Time/space trade-offs for reversible computation," *SIAM J. Comput.*, 1989.
[4] R. J. Lipton and R. E. Tarjan, "Applications of a Planar Separator Theorem," *SIAM J. Comput.*, vol. 9, no. 3, pp. 615–627, 1980.
[5] S. M. Chan *et al.*, "Hardness of approximation in PSPACE and separation results for pebble games," in *FOCS*, 2015.
[6] M. Saeedi and I. L. Markov, "Synthesis and optimization of reversible circuits - a survey," *ACM Comput. Surv.*, 2013.
[7] A. Zulehner and R. Wille, *Introducing design automation for quantum computing*. Springer, 2020, vol. 11.
[8] G. Meuli *et al.*, "Reversible pebbling game for quantum memory management," in *DATE*, 2019.
[9] D. Bhattacharjee *et al.*, "Reversible pebble games for reducing qubits in hierarchical quantum circuit synthesis," in *ISMVL*, 2019.
[10] G. Meuli *et al.*, "Xor-And-Inverter Graphs for Quantum Compilation," *npj Quantum Inf.*, 2022.
[11] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2016.
[12] E. T. Campbell and M. Howard, "Unified framework for magic state distillation and multiqubit gate synthesis with reduced resource cost," *Phys. Rev. A*, 2017.
[13] B. Komarath, J. Sarma, and S. Sawlani, "Pebbling meets coloring: Reversible pebble game on trees," *J. Comput. Syst. Sci.*, 2018.
[14] R. Krenz-Bååth, "Dominator-based algorithms in logic synthesis and verification," Ph.D. dissertation, KTH, 2007.
[15] D. Harel, "A linear time algorithm for finding dominators in flow graphs and related problems," in *ACM STOC*, 1985.
[16] T. Lengauer and R. E. Tarjan, "A fast algorithm for finding dominators in a flowgraph," *ACM Trans. Program. Lang. Syst.*, 1979.
[17] K. D. Cooper *et al.*, "A simple, fast dominance algorithm," *S:PE*, 2001.
[18] C. Gidney, "Halving the cost of quantum addition," *Quantum*, 2017.