A Divide-And-Conquer Pebbling Strategy for Oracle Synthesis in Quantum Computing

Kezhen Zhang, Riling Li, and Mingsheng Ying

***Abstract*—Quantum oracles are quantum circuits that implement classical Boolean functions, frequently used as independent black boxes in quantum algorithm design. The hierarchical reversible logic synthesis approach, a scalable method for large-scale oracle synthesis, relies on ancilla qubits to store intermediate results. The sequence in which these results are computed and uncomputed, governed by the reversible pebble game, greatly affects both qubit usage and circuit gate count. This article introduces a novel divide-and-conquer pebbling strategy that reduces qubit count while maintaining a reasonable circuit gate count. Experimental results show up to a 90% reduction in qubits compared to the most commonly-used strategy, with an average increase of 3 to 4 times in quantum operations, all achieved within a reasonable runtime. In qubit-constrained tasks, our method achieves over 90% reduction in T-count. Additionally, we have incorporated our strategy into the hierarchical synthesis method and implemented it as a potential synthesizer for Qiskit, with experiments demonstrating that it outperforms the default synthesizer. The proposed approach shows promise in optimizing both qubit usage and circuit gate count for large-scale oracle synthesis problems.**

***Index Terms*—Quantum computing, quantum oracle, reversible logic synthesis, reversible pebble game.**

I. Introduction

MANY quantum algorithms utilize a quantum circuit to implement a classical function $f : \{0,1\}^m \rightarrow \{0,1\}^n$, often assumed to exist and applied directly. Such a circuit is commonly known as an oracle, and its implementation is regarded as a distinct problem. In the realm of quantum algorithm research, oracles play a pivotal role in circuit design, often comprising the majority of the final circuit [1]. The frequency of oracle invocations can serve as a metric for assessing algorithm complexity [2]. Due to their frequent utilization in quantum circuits, optimizing the overall cost of oracles has become a significant focus. Various synthesis and optimization methods for oracles have been proposed in recent years [1], [3], [4], [5]. In particular, oracle synthesis functionality has been integrated into quantum software platforms, such as Qiskit [6] and *Q*# [7].

The primary challenge in oracle synthesis is the efficient minimization of redundant operations. For instance, Exclusive-or sum-of-products (ESOP)-based synthesis [8] converts the Boolean function into ESOP form and proceeds to execute ESOP minimization, a computationally challenging procedure. An effective strategy for improving efficiency is the utilization of the hierarchical approach [9], whereby the target function is transformed into a composition of look-up tables (LUTs), each representing a simpler function. Subsequently, ESOP-based synthesis is applied to these reduced-scale LUTs, after which the resulting circuits are recombined to generate the final circuit.

The main limitation of the hierarchical method is its reliance on additional ancillae, helper qubits initialized in the state $|0\rangle$, to preserve the outputs of the LUTs, also known as intermediate results. In a given oracle synthesis task, the ancilla requirement is mainly determined by two factors. First, the sizes of the LUTs play a crucial role. Generally speaking, smaller LUTs facilitate the application of more accurate but computationally complex ESOP minimization algorithms, resulting in shorter circuits. However, this also leads to a larger number of LUTs, thereby increasing the ancilla requirement. Second, the order in which LUTs are computed and uncomputed also influences the ancilla requirement. Drawing parallels with the register allocation problem in classical compilation, one can reduce the required number of ancilla qubits by eagerly cleaning up no-longer-needed results and temporarily uncomputing variables that will not be utilized in the near future. The quantum counterpart of this process is akin to playing the reversible pebble game [10], [11].

By finding a strategy that solves the reversible pebble game with the minimum qubit requirement, we can not only conserve the limited qubit resources on a quantum computer but also optimize the size of the LUTs when the available number of qubits is known. This optimization can lead to a decrease in both the number of qubits and circuit gate count if done properly. However, such a strategy is known to be computationally hard [12]. While a SAT-based exact synthesis strategy has been proposed to tackle this challenge [13], its excessively long runtime limits its applicability to large-scale cases. Alternatively, the reversible pebble game heuristic (RPGH) strategy attempts to provide a suboptimal solution using heuristic methods. However, it often results in a significant increase in circuit gate count, ranging from 10 to over 100 times [14]. Presently, the most prevalent approach to addressing the reversible pebble game involves employing the eager clean-up method [5], a straightforward strategy that reduces qubit usage without introducing additional operations.

Received 24 October 2024; revised 18 February 2025; accepted 6 May 2025. Date of publication 15 May 2025; date of current version 24 November 2025. This work was supported in part by the Key Research Program of the Chinese Academy of Sciences under Grant ZDRW-XX-2022-1, and in part by the Beijing Nova Program under Grant 20230484385. This article was recommended by Associate Editor B. Schmitt. *(Corresponding author: Riling Li.)*

Kezhen Zhang and Riling Li are with the Institute of Software, Chinese Academy of Sciences, Beijing 100190, China (e-mail: lirl@ios.ac.cn).

Mingsheng Ying is with the Centre for Quantum Software, University of Technology Sydney, Sydney, NSW 2006, Australia.

Digital Object Identifier 10.1109/TCAD.2025.3570608

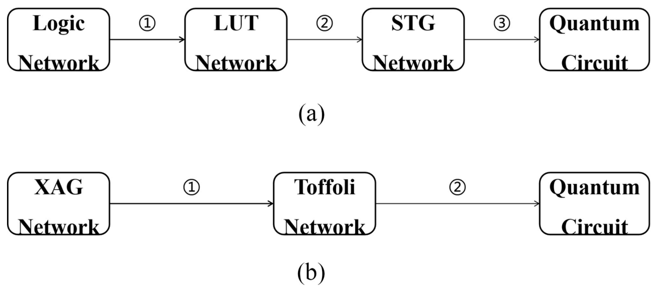

Fig. 1. Overview of the LHRS framework and the XAG-based approach. (a) LHRS framework. Step 1: LUT mapping; Step 2: Reversible pebble game; Step 3: STG synthesis. (b) XAG-based approach. Step 1: XAG-adapted reversible pebble game; Step 2: Toffoli decomposition.

Contributions of the Article: In this article, we introduce a novel *divide-and-conquer-based strategy* for the reversible pebble game. We have devised a recursive approach to solve a specific form of the reversible pebble game, which we then generalized into a strategy applicable to common cases. Our new strategy significantly reduces the required qubit count compared to previous approaches while still maintaining reasonable circuit gate counts. We evaluate the effectiveness of our strategy within the LUT-based Hierarchical Reversible Logic Synthesis (LHRS) framework [9], outlined in Fig. 1(a). Experimental results on the EPFL combinational benchmark [15] demonstrate that integrating our strategy into the second step of LHRS achieves up to a 90% reduction in qubit count compared to the original approach, with a runtime of no more than 2 min and a quantum operation increase of no more than ten times. We further validate our strategy by testing it within an Xor-And-Inverter Graph (XAG)-based logic synthesis approach [16] [outlined in Fig. 1(b)], which shares similarities with the LHRS algorithm, and observe comparable performance improvements. Additionally, our strategy facilitates a more optimized selection of parameters during the LUT mapping step for qubit-constrained tasks, resulting in a reduction of up to 90% in the T-count of the final circuit. Furthermore, we have integrated our strategy as a potential synthesizer for Qiskit's PhaseOracle and BooleanExpression classes, with experiments showing that it outperforms Qiskit's default synthesizer in both CNOT count and depth. Our strategy exhibits the capability to address large-scale problems that are beyond the capability of SAT-based methods.

II. Preliminaries

In this section, we provide a brief introduction to the LHRS framework and explain how the reversible pebble game reduces the number of ancilla qubits required by the algorithm.

A. Single-Target Gate Synthesis

In the field of reversible logic synthesis, single-target gates (STGs) serve as fundamental components for constructing reversible circuits. An STG is denoted as $T_c(\{x_1, \dots, x_k\}, x_{k+1})$, where $x_1, \dots, x_k$ represent the control lines, x_{k+1} denotes the target line, and $c : \{0, 1\}^k \to \{0, 1\}$ denotes the control function. It implements the reversible function $f : \{0, 1\}^{k+1} \to \{0, 1\}^{k+1}$ that preserves the values of the control lines ($f(x_i) = x_i$ for $1 \le i \le k$) while modifying the target line according to the control function $f(x_{k+1}) = x_{k+1} \oplus c(x_1, \dots, x_n)$. It is common practice to convert a reversible circuit into a cascade of STGs before applying STG synthesis methods to decompose them into basic gates.

Multiple-controlled Toffoli gates are STGs whose control functions can be expressed with a single product term. For an STG with a control function $c = c_1 \oplus c_2 \oplus \cdots \oplus c_l$ where $c_1, \dots, c_l$ are product terms, ESOP-based STG synthesis decomposes the STG into a cascade of multiple-controlled Toffoli gates

$$T_c(X, t) \ \to T_{c_1}(X, t) \circ T_{c_2}(X, t) \circ \cdots \circ T_{c_l}(X, t) \tag{1}$$

where $\circ$ denotes the concatenation of gates, and $T_{c_1}(X, t), \dots, T_{c_l}(X, t)$ are multiple-controlled Toffoli gates whose decomposition into basic gates has been extensively studied [17], [18]. However, the expression $c = c_1 \oplus c_2 \oplus \cdots \oplus c_l$ for a given function c, which is called the ESOP form of c, is not unique. Therefore, ESOP minimization algorithms are required to ensure the nonredundancy of the final results. Since the exact minimization of ESOP expressions [19] is computationally difficult and time-consuming, heuristic methods, such as Pseudo-Kronecker Expression (PKRM) [20] and exorcism [21], as well as pseudo-exact approaches that combine exact and heuristic methods [22], are commonly used to balance effectiveness and efficiency.

B. Hierarchical Reversible Logic Synthesis

Hierarchical reversible logic synthesis methods typically rely on logic networks, a data structure that represents a Boolean function using a directed acyclic graph (DAG). In these networks, function inputs, function outputs, and Boolean operations are depicted as vertices, while computation dependencies are described by directed edges. The term "hierarchical" signifies an approach wherein the original network is subdivided into smaller subnetworks, with each subnetwork subsequently mapped into circuits.

The LHRS [9] framework employs LUT mapping [23], [24] to partition the original network into LUTs. This framework typically operates with logic networks defined over a restricted gate basis to represent the input function. For example, the XAG is a widely used type of logic network in this context, where Boolean operations are confined to $\{\wedge, \oplus, \neg\}$. Such networks are commonly employed in related research endeavors [16], [25]. Obtaining such a network for a given Boolean expression is straightforward, after which the LUT mapping procedure is applied to map the original network into subnetworks, i.e., LUTs in this case. Fig. 2(a) illustrates an example of the XAG representation of a two-bit half adder, where dashed arrows represent negation. Fig. 2(b) depicts the corresponding LUT network obtained after LUT mapping.

After obtaining an LUT network, a reversible circuit constructed from STGs can be generated by mapping each LUT into an STG, with an ancilla qubit serving as the target line. Fig. 2(c) and (d) depict two possible STG networks derived from Fig. 2(b). In these figures, dashed lines indicate that the line is not an input to the STG. It's important to note that an

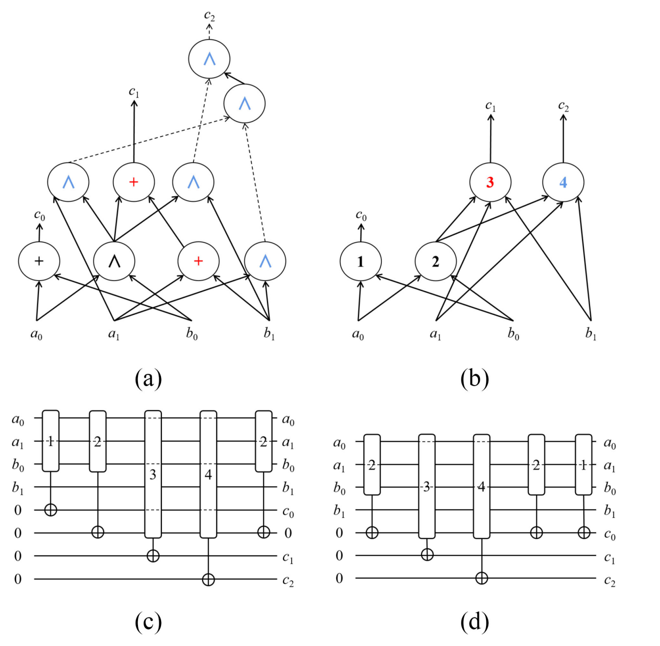

Fig. 2. Example of the LHRS flow working on a two-bit half adder. (a) XAG representation. (b) Corresponding LUT network. (c) STG network mapped from the LUT network. (d) Equivalent STG network with fewer ancilla qubits.

additional STG is utilized to restore the target line of STG 2 to its initial state, a process known as "uncomputation" [26], which is necessary due to the inherent nature of quantum computing. Finally, each STG is transformed into basic gates through the application of STG synthesis algorithms.

The XAG-based approach [16] follows a framework similar to the LHRS algorithm but is specifically tailored for XAG networks. It leverages the fact that each $\oplus$ (exclusive-or) operation in an XAG can be implemented using an in-place CNOT gate, allowing intermediate results to be stored on one of the already-allocated qubits. The reversible pebble game is applied to an abstract graph derived from the original XAG, where only AND nodes are retained. Subsequently, each AND node is computed or uncomputed using a Toffoli gate, which is then decomposed into basic quantum gates.

C. Reversible Pebble Game and Existing Strategies

The contrast between Fig. 2(c) and (d) highlights the potential to reduce the necessary number of ancilla qubits by rearranging the order of the STGs. Addressing this issue involves engaging in the reversible pebble game.

The reversible pebble game is played on a DAG $G = (V, E)$, where each vertex $v \in V$ corresponds to an input, an output or an intermediate result and directed edges E represent computation dependency. An example is shown in Fig. 3. The bottom vertices x_1, x_2, x_3, x_4 are the primary inputs, whose data are available throughout the process. Such vertices are sometimes omitted in the DAG for simplicity. Vertices 7 and 8 marked by y_1, y_2 are primary outputs that we need to acquire.

In the reversible pebble game, when we intend to compute or uncompute an intermediate result represented by a vertex v, we use the terms "pebble v" or "unpebble v".

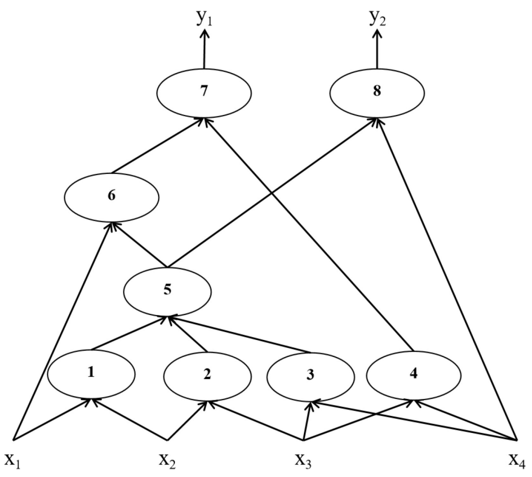

Fig. 3. Example of a reversible pebble game.

Definition 1 (Reversible Pebbling Configuration): A reversible pebbling configuration of DAG G is a set $P \subseteq V$, which is the set of all pebbled vertices at a given step.

Definition 2 (Reversible Pebbling): Let O denote the set of all primary outputs of DAG G. A reversible pebbling of G is a sequence of reversible pebbling configurations $P = (P_1, P_2, \ldots, P_m)$ where $P_1 = \emptyset$ and $P_m = O$, and for each $2 \le i \le m$ we have

1) $P_i = P_{i-1} \cup \{v\}$ or $P_i = P_{i-1} \setminus \{v\}$ for some $v \in V$ and $P_i \neq P_{i-1}$.
2) All in-neighbors of v are in P_{i-1}.

The number of pebbles used in a reversible pebbling P is defined as the maximum size of reversible pebbling configurations in the sequence

$$p = \max_{1 \le i \le m} | P_i |. \tag{2}$$

The reversible pebble game aims at finding the minimum p and the corresponding reversible pebbling P.

Previous research has already obtained various conclusions under certain constraints. For example, the time-optimal solution under a given space bound can be expressed in recursive form for line graphs [27]. Additionally, it has been demonstrated that determining the minimum p along with the reversible pebbling can be achieved in polynomial time for trees with a single primary output [11]. However, these conclusions do not extend to an arbitrary DAG. For a random DAG, the problem can be reformulated as a satisfiability problem (SAT problem), where p acts as a parameter constraining the number of pebbles [13]. A SAT solver is invoked to verify the existence of a solution within p, and the value of p is successively increased until either the minimum value is reached or the time limit is exceeded.

Due to the computational complexity inherent in solving SAT problems, the SAT-based approach may encounter scalability issues. To the best of our knowledge, there are mainly 3 alternative strategies that can be applied to relatively large DAGs in practice.

1) *Bennett Strategy [10]:* This is a straightforward approach that involves pebbling all vertices in topological order, followed by unpebbling all vertices except the

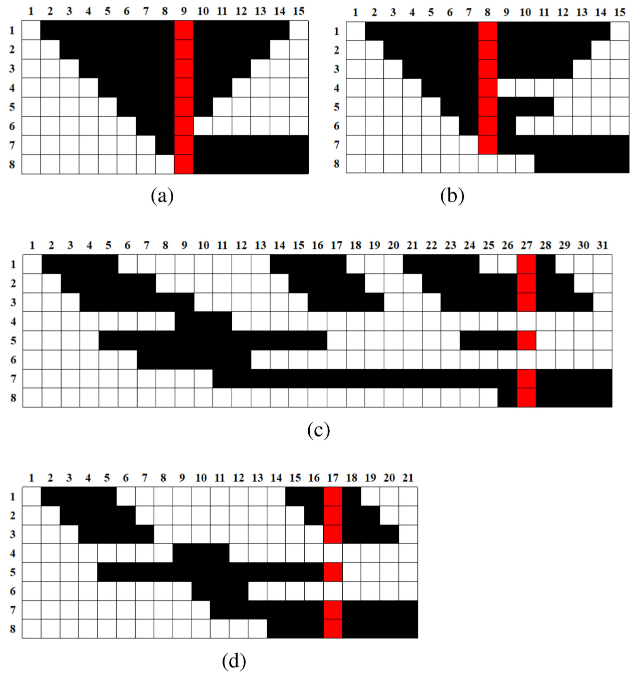

Fig. 4. Different pebbling strategies for the DAG in the example. Each column represents a reversible pebbling configuration at a specific step, with each black square indicating a pebbled vertex. The red columns highlight the steps where the maximum number of pebbles are used. Our strategy uses six pebbles and spans 20 steps. The Bennett strategy and the eager clean-up strategy require more pebbles (8 and 7, respectively), while the RPGH strategy necessitates more steps (30 steps). (a) Bennett strategy. (b) Eager clean-up strategy. (c) RPGH strategy. (d) Our strategy.

primary outputs in reverse topological order. This strategy utilizes $|V|$ pebbles, and it requires the minimum number of steps to complete the pebbling process.

2) *Eager Clean-Up Strategy [5]:* This strategy is similar to the Bennett strategy but focuses on unpebbling intermediate results as soon as possible. In this approach, when a primary output with no immediate successor is pebbled, its direct predecessors are recursively examined to determine if they have unexploited out-edges. If none are found, they are unpebbled. Despite its simplicity, the eager clean-up strategy does not increase the number of steps compared to the Bennett strategy. As a result, it offers an efficient approach for pebbling DAGs, particularly when implemented in algorithms, such as the LHRS algorithm.

3) *RPGH Strategy [14]:* The primary outputs are arranged in descending order based on the sizes of their transitive fan-in cones, and they are pebbled individually. During each pebbling step, the strategy determines the next move by evaluating vertices using a heuristic function. Furthermore, before initiating each pebbling step, it endeavors to reclaim free ancilla qubits by attempting to unpebble previous vertices in a heuristic manner.

The reversible pebblings of the strategies above are depicted in Fig. 4. It is noteworthy that the RPGH strategy exhibits a number of unnecessary steps (from step 16 to step 23).

D. Utilization of Free Ancillae

Both the LHRS algorithm and the XAG-based approach require a significant number of ancilla qubits. When the intermediate result stored on an ancilla qubit is uncomputed, the qubit is considered clean and can be reused for other optimization methods until it is needed for another intermediate result. Below, we introduce two optimization techniques that leverage the utilization of ancilla qubits.

Ancillae-Aided Toffoli Decomposition [18]: This method involves computing the AND of some control qubits and storing the result on ancilla qubits, which are then used as the control qubits for subsequent multiple-controlled Toffoli gates. This effectively breaks down a multiple-controlled Toffoli gate into smaller gates with fewer controls, reducing the total number of quantum gates after decomposition. This approach aligns well with ESOP-based STG synthesis, as it naturally results in a cascade of multiple-controlled Toffoli gates.

Best-Fit Mapping [28]: This technique is based on the observation that, after LUT mapping, each LUT in the network corresponds to a subnetwork of the initial logic network, referred to as cells, which preserve the logic network's structure. For example, in Fig. 2(a), the cells corresponding to LUT 3 and LUT 4 in Fig. 2(b) are highlighted in the same color. Each cell contains a vertex representing its output, which also corresponds to the output of the LUT. The direct predecessors of the LUT in Fig. 2(b) serve as the input vertices of the cells in Fig. 2(a). This enables a second round of LUT mapping with the assistance of clean ancilla qubits, where each cell can be further divided into smaller LUTs, thereby reducing the cost of subsequent synthesis.

III. Divide-and-Conquer Strategy

In this section, we introduce the main idea of our strategy and present an outline of the algorithm implementation.

A. Main Idea

Consider the simplest situation where the DAG is shaped as a chain (see Fig. 5). It is evident that when the second-to-last vertex is pebbled, simply reversing all previous steps after pebbling the last vertex completes a valid reversible pebbling. Consequently, the problem is reduced to finding the longest chain whose tail can be pebbled using $p - 1$ pebbles.

When $p - 1 = 2$, it is easy to verify that a chain of length 3 is the maximum attainable, as depicted in Fig. 5(a) (primary inputs are omitted). When $p - 1 = 3$, we utilize the previous conclusion to pebble a chain of length 7, as illustrated in Fig. 5(b). It is noteworthy that the sections marked by rectangles are identical to that in Fig. 5(a). By recursively applying the same steps, a chain of length 2^{p-1} can be pebbled with p pebbles. Thus, a chain of length n can be pebbled with $\lceil \log_2(n) \rceil + 1$ pebbles, which proves to be optimal [10].

The process described above is a special case of the recursion on line graphs [27]. The key to the strategy lies in identifying a checkpoint within the chain and then subdividing the problem into subproblems. This concept serves as the inspiration for adapting the divide-and-conquer strategy to an arbitrary DAG, as outlined in Section III-B.

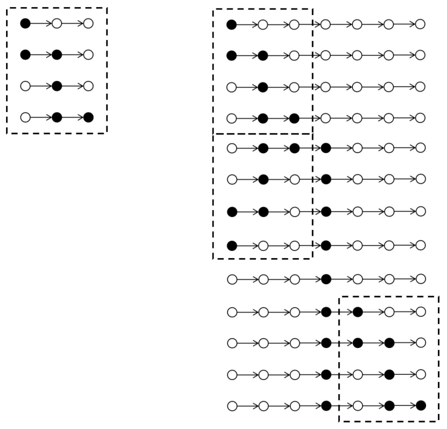

Fig. 5. Pebble the possibly longest chain with a fixed number of pebbles. (a) $p-1=2$. (b) $p-1=3$.

Fig. 4(d) illustrates the application of our strategy to the DAG depicted in Fig. 3. The fundamental approach involves designating vertex 5 as a checkpoint. When vertex 5 is pebbled, its predecessors are immediately unpebbled, and vertex 5 is treated as a new starting point for the second stage of the pebble game. Both stages are managed using the eager clean-up strategy. In comparison to the RPGH strategy, our approach yields the same number of pebbles but requires fewer steps. Furthermore, as the size of the DAG increases, the heuristic nature of the RPGH strategy introduces unpredictability, whereas our strategy ensures controlled unpebbling. Therefore, we anticipate that our strategy will demonstrate greater competence and efficiency when dealing with large-scale problems.

B. Algorithm Outline

Algorithm 1 outlines the main body of the divide-and-conquer pebbling strategy. The function pebble serves as the primary entry point of the algorithm (line 1), taking four parameters as inputs.

1) G represents the DAG this procedure operates on. For simplicity we use the subset $V' \subseteq V$ to denote its induced subgraph $G[V'] = (V', E')$, where $E' = \{(i,j) \in E \mid i,j \in V'\}$.
2) *sta* denotes the set of vertices that are initially pebbled and remain pebbled throughout the procedure. For a given reversible pebble game problem, *sta* is initialized as the set of all primary inputs.
3) *tgt* represents the set of target vertices that we aim to pebble or unpebble in this procedure. For a given reversible pebble game problem, *tgt* is initialized as the set of all primary outputs.
4) *rev* is a boolean variable that defines the objective of this procedure. If *rev* = *true*, we intend to unpebble all vertices in *tgt*, meaning that initially, the vertices in $sta \cup tgt$ are pebbled, and after this procedure, only the vertices in *sta* remain pebbled. Otherwise, we start with all vertices in *sta* and conclude with $sta \cup tgt$.

In the function pebble, we first partition G into connected subgraphs and identify the "live" nodes that contribute to at least one *tgt* vertex. Subsequently, we invoke the DivideAndConquer function on each of these "live" subgraphs (lines 2–4). This preprocessing ensures that the subgraph passed to DivideAndConquer is semiconnected and each nontarget vertex has at least one successor.

The core of the strategy lies within the DivideAndConquer function (lines 5–16). When the size of the input subgraph is deemed sufficiently small, we apply the eager clean-up strategy to it (lines 6–7). Details regarding how we define "small" are elucidated in Section IV-B. Alternatively, if the subgraph is larger, we endeavor to identify a set S of vertices that serves as the checkpoint in the current problem (line 8). The methodology for identifying the set S is expounded upon Section IV-A. Subsequently, we further partition the problem using S. In the initial stage, the set S together with $A \cap tgt$ is pebbled through A starting with *sta* (line 14). Following this, we attempt to pebble the *tgt* vertices in set B with $S \cup sta$ as the new starting point (line 15). Finally, we unpebble the set S except for the *tgt* vertices within it (line 16). Note that the set $A \cup S \cap tgt$ has been pebbled in the initial stage and does not require cleaning, hence it constitutes part of the *sta* parameter of the subproblem according to our definition. The reverse division of this process is described between lines 10 and 12.

Ideally, the number of pebbles used by the divide-and-conquer strategy is approximately the sum of the sizes (i.e., the number of elements, where each element is a vertex) of all the checkpoint sets and the *tgt* set, which is dependent on the geometric features of the initial DAG. It is worth noting that we could have executed our strategy on each transitive fan-in cone of the primary outputs, similar to the RPGH strategy, potentially resulting in a larger decrease in the number of pebbles. However, our strategy operates on each connected subgraph instead (line 2). This decision is made to preserve the DAG's ability to leverage shared parts among different cones.

IV. Implementation Details

In this section we decsribe the unexplained details in the implementation of Algorithm 1.

A. Checkpoint Identification

The set S selected as the checkpoint must satisfy certain criteria. Primarily, the vertices in S and *sta* should collectively suffice to pebble the vertices in set B. Additionally, to ensure the effectiveness of the split stages, we aim to maintain a clear separation between the induced subgraphs $G[A]$ and $G[B]$, with $G[S]$ acting as the divider. Furthermore, since the vertices in S remain pebbled during the second stage of the divided subproblems, the size of S significantly impacts the final cost. Therefore, we strive to minimize the size of the set S.

Motivated by a greedy algorithm for the vertex separator problem [29], we employ Algorithm 2 to identify the checkpoint set S. The algorithm iteratively inserts vertices into the initially empty subgraph $G[A]$, ensuring its size gradually approaches half of G, with each step prioritizing the vertex

Algorithm 1: Divide-and-Conquer Pebbling

```
1  Function Pebble(G=(V, E), sta, tgt, rev):
2      forall connected subgraph of G do
3          $liv \leftarrow \{v \in subgraph \mid \exists t \in tgt$, there is a path from $v$ to $t\}$;
4          $DivideAndConquer(liv, sta, tgt \cap liv, rev)$;
5  Function DivideAndConquer(G=(V, E), sta, tgt, rev):
6      if $|V|$ is small then
7          use eager clean-up strategy on G;
8      $(A, B, S) \leftarrow FindCheckpoint(G, sta, tgt)$;
9      if rev then
10         $Pebble((A \cup S) \setminus tgt, sta \cup (A \cup S \cap tgt), S \setminus tgt, false)$;
11         $Pebble(B, S \cup sta, B \cap tgt, true)$;
12         $Pebble(A \cup S, sta, S \cup (A \cap tgt), true)$;
13     else
14         $Pebble(A \cup S, sta, S \cup (A \cap tgt), false)$;
15         $Pebble(B, S \cup sta, B \cap tgt, false)$;
16         $Pebble((A \cup S) \setminus tgt, sta \cup (A \cup S \cap tgt), S \setminus tgt, true)$;
```

Algorithm 2: Identification of Checkpoint Set

```
1  Function FindCheckpoint(G=(V, E), sta, tgt):
2      $A \leftarrow \emptyset, B \leftarrow V$;
3      $C \leftarrow candidate_G(A \cup sta)$;
4      $S \leftarrow fanin_G(B)$;
5      while $|A| \leq |V|/2$ do
6          let $i \in C$ s.t. $i$ minimizes $|fanin_G(B \setminus \{i\})|$;
7          move $i$ from $B$ to $A$;
8          update $C$ and $S$;
9      $A \leftarrow A \setminus S$;
10     return $(A, B, S)$;
```

that minimizes the separator S (lines 5–8). The *candidate* set and *fanin* set are defined as

$$\text{candidate}_{G=(V,E)}(A) = \{v \in V \mid \forall (u, v) \in E, u \in A\} \quad (3)$$

$$\text{fanin}_{G=(V,E)}(B) = \{u \in V \setminus B \mid \exists v \in B, (u, v) \in E\} \quad (4)$$

where *candidate* represents the set of vertices that can be pebbled in the subsequent step, while *fanin* comprises all vertices required to compute $G[B]$. This configuration guarantees that $S \cup sta$ in Algorithm 1 encompasses all essential predecessors for handling the second stage.

B. Stopping Condition

For an input DAG $G = (V, E)$, the pebble number of the divide-and-conquer strategy is approximately $|S| + \max(p_e(G[A]), p_e(G[B]))$, where p_e represent the number of pebbles used by eager clean-up strategy. Since the sizes of A and B are around $|V|/2$, as the size of V decreases, the advantage gained from applying divide-and-conquer strategy over eager clean-up strategy diminishes. On the other hand, deeper recursion entails additional computation and uncomputation steps. As a result, we anticipate the recursion to halt once $|V|$ reaches a sufficiently small value. In the remainder of this section we present two methods to determine the stopping condition of the recursion.

A Threshold-Based Method: A straightforward approach is to establish a threshold for the size of the input graph G, beyond which the recursion stops and eager clean-up strategy is employed. For instance, the threshold might be set to be $1/10$ of the size of the input DAG. Thus, when processing a DAG with 100 vertices, the recursion terminates once the size of the divided subgraph falls below 10. However, this method has drawbacks. First, it may lead to numerous unnecessary iterations, especially when G is split into multiple parallel subproblems. Second, the threshold does not offer direct control over the pebble number of the final result.

Calculating Qubit Allocation and Reclamation: An alternative approach involves considering the problem from the perspective of qubit allocation and reclamation. During the reversible pebble game, when an unpebbling move occurs, it signifies that a qubit is reclaimed and available for future allocation. Reusing such reclaimed qubits does not increase the overall ancilla requirement. Therefore, by monitoring the number of reclaimed qubits, denoted as p_r, and comparing it with the size of the input DAG G, the algorithm can determine when $p_e(G) \leq |V| \leq p_r$. Applying eager clean-up strategy under such circumstances minimizes unnecessary pebbling steps without escalating the total pebble number. Moreover, by initializing p_r to a value greater than 0, we effectively allocate a predetermined number of qubits for use in the synthesis task. In such cases, Algorithm 1 attempts to find a reversible pebbling with a lower pebble count than p_r which serves as a limit. Should this prove unsuccessful, it returns a sequence with the lowest pebble count it can find. Our strategy reverts to eager clean-up strategy when the limit is set to a number greater than $|V|$.

When a qubit allocation occurs, we have the option to either reuse a reclaimed qubit or allocate a new one. This aspect of the second approach can be managed using heuristic algorithms like [30], which take factors, such as qubit connectivity into account. Currently, our priority is to minimize the ancilla requirement, so we opt to allocate reclaimed qubits whenever possible.

The second approach provides a more precise control over the final pebble count compared to the threshold-based method. However, there may be instances where users are unable to determine a suitable limit beforehand. Consequently, we adopt a hybrid approach where Algorithm 1 terminates the recursion when either the size of the input graph falls below the predetermined threshold or sufficient qubits are reclaimed to directly apply the eager clean-up strategy to the input graph.

V. Optimization of Qubit Resource Utilization

In this section, we discuss strategies to maximize the utilization of available qubit resources.

A. Working Against Qubit Constraints

Rather than focusing on identifying the minimum qubit requirement for a given circuit, quantum circuit synthesis

TABLE I
COMPARISON OF PEBBLING STRATEGIES ON THE ISCAS85 BENCHMARKS

Benchmark	PI/PO	Bennett	Eager clean-up		SAT-based		RPGH			Divide-and-conquer			$\%\Delta\mathbf{p}_s$	$\%\Delta\mathbf{p}_r$
		pebbles	pebbles	steps	pebbles	steps	pebbles	steps	time/s	pebbles	steps	time/s		
c432	36/7	169	157	342	56	1009	81	1642	0.28	40	1042	0.02	28.6	50.6
c499	41/32	160	102	352	79	774	73	10114	0.79	26	1006	0.03	67.1	64.4
c880	60/26	303	123	630	85	1556	74	2154	0.22	64	1176	0.03	24.7	13.5
c1355	41/32	472	321	976	FAIL		202	28864	8.70	37	2730	0.07	N/A	81.7
c1908	33/25	389	297	803	FAIL		149	14149	2.88	57	2461	0.08	N/A	61.7
c2670	157/63	698	523	1415	FAIL		189	3487	1.12	71	4327	0.28	N/A	62.4
c3540	50/22	1026	954	2064	FAIL		498	11104	17.50	135	6174	0.38	N/A	72.9
c5315	178/123	1702	688	3475	FAIL		277	22843	15.47	211	9165	0.90	N/A	23.8
c6288	32/32	2306	2251	4643	FAIL		MEMORY OUT			207	41385	2.97	N/A	N/A
c7552	207/108	1308	797	2654	FAIL		246	12332	7.55	165	7662	0.75	N/A	32.9

Average percentage of decrease in pebbles compared to the SAT-based approach: 40.1
Average percentage of decrease in pebbles compared to the RPGH strategy: 51.6

typically operates under the constraint that a synthesis task must be completed within a specified qubit limit. In the LHRS algorithm, the determination of the qubit requirement hinges on two factors: 1) the cut size of the LUT mapping process and 2) the selection of strategy for the reversible pebble game. The choice of algorithm for LUT. mapping also plays a role but is not discussed in this context.

Smaller Cut Sizes Yield Better Synthesis Results: The cut size in the LUT mapping process, indicating the maximum number of inputs to each LUT, significantly influences both the qubit requirement and the quality of the final result. Larger cut sizes tend to generate fewer but larger-sized LUTs (measured by the number of inputs) that lead to longer runtime and potentially inferior results in the subsequent STG synthesis stage, with fewer qubits required to store the intermediate results. Conversely, smaller cut sizes result in a higher qubit requirement but enable the application of more sophisticated algorithms that yield better results. Furthermore, for LUTs with a sufficiently small number of inputs (typically fewer than 4), it is possible to explore a database of precomputed near-optimal circuits, which produces satisfactory results in a relatively short amount of time.

Determining Cut Sizes in LHRS: The LHRS algorithm addresses this tradeoff through an incremental approach. It initiates LUT mapping with a small cut size and employs eager clean-up strategy to determine the required number of qubits. If the qubit requirement exceeds the specified limit, the algorithm iteratively increases the cut size until a suitable configuration is found where the required qubits meet the constraint.

Reducing the Cut Size With Better Pebbling Strategy: In the preceding sections, we introduced our innovative divide-and-conquer strategy, which reduces the number of qubits required for a given LUT network compared to the eager clean-up strategy. As a result, when determining the appropriate cut size under a specified qubit constraint through an incremental process, employing the divide-and-conquer strategy instead of the eager clean-up strategy allows for an earlier conclusion, yielding a smaller cut size that benefits the subsequent STG synthesis procedure. Although our strategy results in an increased number of computation and uncomputation steps, each corresponding to an STG in the LHRS algorithm, we will demonstrate in Section VI that the reduction in circuit gate count resulting from the smaller LUT sizes outweighs the increase in the number of steps.

B. Best-fit Mapping With Divide-and-Conquer Pebbling Strategy

Reducing Cut Sizes With Subsequent LUT-Mapping: As discussed in Section II-D, the best-fit mapping technique applies a second round of LUT mapping to each cell of the original logic network. This approach further reduces LUT sizes, leading to a more compact circuit.

Combining Divide-and-Conquer Strategy With Best-Fit Mapping: To avoid increasing the total ancilla qubit requirements, the second round of LUT mapping is restricted to qubits already allocated during the first round, thereby setting a constraint for qubits. Since each cell has only one output node, the eager clean-up strategy does not help in reducing qubit usage in such cases. However, using the divide-and-conquer strategy, the cells can be decomposed into even smaller LUTs, enabling more efficient near-optimal mapping throughout the circuit.

VI. RESULTS

We implemented our strategy as part of the LHRS algorithm [9] in C++ with caterpillar,[1] one of the EPFL logic synthesis libraries [31]. Experiments were run on an Intel Core i7-12700 CPU with 2.10 GHz and 16 GB RAM running Windows 11 OS.

A. Comparison of Pebbling Strategies

We evaluate the performance of various pebbling strategies on the ISCAS85 benchmark, and the results are presented in Table I. The "PI/PO" column denotes the number of primary inputs and primary outputs for each benchmark. The metrics "pebbles" and "steps" represent the number of pebbles (excluding PIs and POs) used and the number of steps in the resulting reversible pebbling, respectively. The $\%\Delta p_s$ and $\%\Delta p_s$ columns indicate the percentage reduction in pebbles compared to the SAT-based approach and the RPGH strategy, respectively.

[1]https://github.com/gmeuli/caterpillar

TABLE II
SYNTHESIS RESULTS OF THE LHRS ALGORITHM ON THE EPFL BENCHMARKS

Benchmark	PI/PO	k	Bennett	Eager clean-up				Divide-and-conquer					%Δq	×s	×T
			qubits	qubits	STGs	T-count	t/s	qubits	STGs	t_p/s	T-count	t/s			
Adder	256/129	6	448	448	255	18393	0.01	400	365	0.09	33271	0.10	10.7(76.2)	1.43	1.81
		10	416	416	191	105221	0.05	401	221	0.07	138447	0.11	3.6(48.4)	1.16	1.32
		15	403	403	165	2029778	1.12	400	171	0.04	2171890	1.20	0.7(16.7)	1.02	1.07
Barrel shifter	135/128	6	647	647	896	50944	0.02	325	1322	0.21	78208	0.24	49.8(83.9)	1.48	1.54
		10	647	647	896	50944	0.02	325	1322	0.21	78208	0.24	49.8(83.9)	1.48	1.54
		15	645	645	892	178488	0.25	314	1384	0.27	226496	0.53	51.3(86.6)	1.55	1.27
Max	512/130	6	1257	1257	1359	108312	0.07	838	5397	4.93	534148	5.08	33.3(68.1)	3.97	4.93
		10	930	930	705	419255	0.19	707	2217	1.61	1448399	2.00	24.0(77.4)	3.14	3.45
		15	846	846	537	2366269	3.77	684	1739	0.98	8690883	5.75	19.1(79.4)	3.24	3.67
Sine	24/25	6	1608	1607	3143	129987	0.11	383	8179	0.87	348731	1.03	76.2(78.6)	2.60	2.68
		10	878	878	1683	1145748	2.76	220	5715	0.49	2823328	3.68	74.9(79.4)	3.40	2.46
		15	723	723	1373	22210982	48.06	187	4891	0.38	97269496	59.72	74.1(79.5)	3.56	4.38
Square	64/128	6	4196	4196	8134	216716	0.13	354	78328	24.08	2073754	24.71	91.6(96.0)	9.63	9.57
		10	3793	3793	7328	751223	0.63	350	72604	22.45	6756989	24.16	90.8(95.6)	9.91	8.99
		15	3001	3001	5744	18258675	63.02	322	48922	14.11	178201467	109.28	89.3(95.4)	8.52	9.76
Log2	32/32	6	8156	8156	16216	1082999	0.35	1212	55362	14.71	3309687	15.59	85.1(85.8)	3.41	3.06
		10	3697	3697	7298	5824718	12.82	447	41878	8.18	31062656	25.81	87.9(89.5)	5.74	5.33
		15	3426	3426	6756	325485491	435.06	402	36632	9.64	MEMORY OUT		88.3(89.9)	5.42	N/A
Multiplier	128/128	6	6706	6616	13028	804593	0.26	608	164390	111.36	9684013	113.79	90.8(94.5)	12.62	12.04
		10	4006	3942	7628	3390350	0.93	563	49774	22.21	19057994	25.90	85.7(91.7)	6.53	5.62
		15	3957	3898	7530	40793730	18.68	542	52472	21.29	220852484	79.62	86.1(92.1)	6.97	5.41
Square-root	128/64	6	8332	8332	16344	341529	0.23	842	145202	38.78	3130481	40.11	89.9(92.0)	8.88	9.17
		10	7966	7966	15612	996727	0.45	866	135708	35.85	9383637	38.56	89.1(91.3)	8.69	9.41
		15	7902	7902	15484	96851816	76.18	839	134924	39.41	MEMORY OUT		89.4(91.6)	8.71	N/A
Divisor	128/128	6	24115	12420	47814	792403	0.64	1613	416282	189.25	6969849	192.52	87.0(88.8)	8.71	8.80
		10	23665	12133	46914	1115809	0.84	1487	441808	282.51	9950865	286.70	87.7(89.6)	9.42	8.92
		15	23367	11955	46318	9128933	26.83	1437	408122	325.03	93342195	372.03	88.0(89.9)	8.81	10.22

Average percentage of decrease in qubits: 69.9(82.7)
Average multiplicative factor for the number of steps: 5.56
Average multiplicative factor for T-count: 5.05

The SAT-based approach is implemented using the Z3 solver [32], with time limits set to 30 min for the solver and 60 min for the overall search. Cases marked as "FAIL" indicate that the SAT-based approach failed to produce a valid result within the allotted time. It is worth noting that while the SAT-based approach has the potential to achieve the minimum number of pebbles in reversible pebbling, the results obtained may not be optimal due to hardware limitations and the imposed time constraints, especially given the relatively large size of the benchmarks. Additionally, the Bennett strategy consistently produces the same number of steps as the eager clean-up strategy, with both strategies completing in under 0.2 s. For brevity, these columns are omitted from the table.

Our strategy achieves a relatively low number of pebbles while maintaining a reasonable runtime. In contrast, the SAT-based approach often requires a significantly longer runtime and may fail to produce results for relatively large cases. The RPGH strategy, while capable of reducing the number of pebbles to some extent, incurs a substantial increase in the number of pebbling steps. Our strategy outperforms both the SAT-based approach and the RPGH strategy in terms of the number of pebbles, the number of steps, and runtime across most cases. Specifically, it achieves an average reduction of 40.1% in the number of pebbles compared to the SAT-based approach and an average reduction of 51.6% compared to the RPGH strategy.

B. Large-Scale Benchmark Synthesis

LHRS Algorithm: We evaluate the performance of our strategy on nine arithmetic benchmarks from the EPFL combinational benchmark [15], employing different cut sizes (denoted as k) within the LHRS algorithm. A cut size of 6 is selected as it is a common choice in related research, while 15 is the maximum value that avoids stack smashing during LUT mapping on our device. Additionally, ten serves as an intermediate value. Results are presented in Table II, which includes comparisons with the Bennett strategy and eager clean-up strategy. The STGs are synthesized into multicontrolled Toffoli networks using ESOP-based synthesis, with ESOP minimization achieved through PKRM [20] and a modified version of exorcism where EXORLINK-4 operations are omitted [9]. Multicontrolled Toffoli gates are decomposed into Clifford+T gates using ancilla qubits (see Section II-D). The number of STGs and T gates in the final circuit, listed in Table II, serve as indicators of circuit gate count. Runtime metrics include total runtime t and runtime consumed by computing the pebbling strategy t_p. For clarity, we denote %Δq as the percentage reduction in qubits compared to eager clean-up strategy, ×s as the multiplicative factor for the number of STGs, and ×T as the multiplicative factor for the number of T gates. All experiments are conducted with the qubit limit for the divide-and-conquer strategy, as mentioned in Section IV-B, set to 0, representing the minimum number of qubits achievable by our strategy. Setting this limit to a larger value may help mitigate the impact on circuit gate count, which we will explore in the next section. No post-synthesis optimization is performed in the experiments detailed here.

The standard form of an oracle is $|x\rangle\,|q\rangle \rightarrow |x\rangle\,|q \oplus f(x)\rangle$, where $f : \{0, 1\}^m \rightarrow \{0, 1\}^n$ is the Boolean function of the oracle [33]. Here, $|x\rangle$ represents the m-qubit register containing the input to the oracle, and $|q\rangle$ denotes the n-qubit register where the outputs of f are stored. It is important to note that $|q\rangle$ is not necessarily initialized as $|0\rangle$. Consequently, it is imperative that all outputs of f are not computed on reclaimed qubits during the reversible pebble game, and

TABLE III
SYNTHESIS RESULTS OF THE XAG-BASED APPROACH ON THE EPFL BENCHMARKS

Benchmark	**Eager clean-up**						**Divide-and-conquer**						**%Δa**	**×T**
	ancillae	**T-count**	Δa_6	$\times T_6$	Δa_{15}	$\times T_{15}$	**ancillae**	**T-count**	Δa_6	$\times T_6$	Δa_{15}	$\times T_{15}$		
Adder	510	13377	447	0.73	492	0.01	25	71715	10	0.04	10	0.03	95.1	5.36
Barrel shifter	2954	45808	2570	0.90	2572	0.26	317	173488	255	0.97	266	0.77	89.3	3.79
Max	2481	39207	1866	0.36	2277	0.02	344	179809	148	0.08	302	0.02	86.1	4.59
Sine	5322	75656	3764	0.58	4648	0.00	426	294406	92	0.01	288	0.00	92.0	3.89
Square	17986	257894	13982	1.19	15177	0.01	850	3522120	688	0.19	720	0.02	95.3	13.66
Log2	31989	448616	23897	0.41	28627	0.00	1558	2316090	410	0.01	1220	N/A	95.1	5.16
Multiplier	26785	377972	20425	0.47	23143	0.01	1489	2693236	1137	0.07	1203	0.01	94.4	7.13
Square-root	24617	344645	16477	1.01	16907	0.00	748	4229309	98	0.04	101	N/A	97.0	12.27
Divisor	28736	800982	16572	1.01	17037	0.09	1195	12338564	-162	1.35	14	0.13	95.8	15.40
Average	15709	267129	11111	0.74	12320	0.04	772	2868749	297	0.31	458	0.14	93.3	7.92

- Δa_k denotes the number of additional ancilla qubits required by the XAG-based approach compared to the LHRS algorithm in Table II, where k represents the corresponding cut size.
- $\times T_k$ denotes the multiplicative factor for T-count relative to the LHRS algorithm in Table II, where k represents the corresponding cut size.
- %Δa and ×T represent the percentage reduction in ancilla qubits and the multiplicative factor for T-count, respectively, of the divide-and-conquer strategy compared to the eager clean-up strategy.

each primary output requires a dedicated qubit that cannot be repurposed. Minor adjustments are made in the LHRS algorithm to accommodate this constraint, which ensures that the primary inputs and primary outputs remain unaffected by the reversible pebble game. The "PI/PO" column of Table II indicates the count of primary inputs and primary outputs in each respective benchmark. In the %Δq column, we present the percentage reduction of remaining qubits (excluding PI and PO ones) in brackets.

Our strategy significantly reduces the number of qubits, achieving an average percentage reduction of 69.9% (82.7% excluding PIs and POs) compared to eager clean-up strategy. This comes with an average of 4.56 times more STGs (or 4.05 times more T gates) in the circuit. The Adder benchmark presents the worst result, where the majority of qubits are used to carry inputs and outputs, leaving limited room for optimization. However, in larger cases where the number of inputs and outputs is dwarfed by the number of intermediate results, our strategy saves over 80% of ancilla qubits compared to eager clean-up strategy, with mostly less than ten times the original number of STGs. In comparison, the RPGH strategy typically results in a percentage reduction of around 10% to 20%, accompanied by an increase in the number of STGs by 10 times to over 100 times in most cases [14], and the SAT-based approach is not applicable to inputs of such scale.

The computation time of our strategy, as shown in the "t_p" column, remains mostly consistent regardless of the increase in the cut size for LUT mapping. However, as the cut size is raised to 15, the total synthesis time of the circuit experiences rapid growth. To expedite the synthesis process, synthesized STGs are cached, and "MEMORY OUT" indicates that the synthesis was aborted after exceeding the 16GB memory limit. Additionally, while the number of STGs shows only minor variation, the T-count of the final circuit increases significantly. These observations suggest that minimizing LUT sizes is beneficial for the STG synthesis process, as discussed in Section V-A.

XAG-Based Approach: We also evaluate the performance of our strategy within the XAG-based approach, and the results are presented in Table III. Specifically, we test the algorithm's ability to reduce T-count [16, Algorithms 1 and 3] where 2-controlled Toffoli gates are decomposed into 7 T gates. Similar to the experiments within the LHRS framework, the number of ancilla qubits (i.e., qubits that do not hold inputs or outputs) and the number of T gates are used as metrics to quantify the cost of the final circuit. For clarity, %Δa and ×T denote the percentage reduction in ancilla qubits and the multiplicative factor for T-count, respectively, achieved by the divide-and-conquer strategy compared to the eager clean-up strategy. Additionally, to compare with the LHRS algorithm, we include the number of additional ancilla qubits required by the XAG-based approach in the Δa_k columns and the multiplicative factor for T-count relative to the LHRS results in the $\times T_k$ columns, where k represents the corresponding cut size in the LHRS framework.

The experiments with the XAG-based approach further demonstrate that the divide-and-conquer strategy significantly reduces the required number of ancilla qubits. Additionally, a comparison with the results of the LHRS algorithm reveals that the XAG-based approach generally achieves better results in terms of the T-count of the final circuit. This advantage is particularly pronounced when the cut size of the LHRS algorithm is set to 15, where the final T-count of the XAG-based approach is less than 1% of that of the LHRS algorithm for benchmarks, such as Sine and Log2. However, this improvement comes at the cost of a significantly higher number of ancilla qubits. When using the eager clean-up strategy, the XAG-based approach requires an average of over 10 000 extra ancilla qubits. This is primarily due to the absence of the LUT-mapping stage in the XAG-based approach, where multiple vertices in the original XAG are fused into a single LUT node, thereby reducing the overall qubit requirement. This drawback may limit the practical applicability of the approach.

By applying the divide-and-conquer strategy instead of the eager clean-up strategy, the demand for additional ancilla qubits is significantly mitigated. With this strategy, the XAG-based approach requires an average of no more than 400 extra ancilla qubits, substantially lowering the threshold for its practical application.

C. *Qubit-Constrained Synthesis*

In the previous section, we demonstrated our strategy's effectiveness in reducing qubit usage when the LUT network

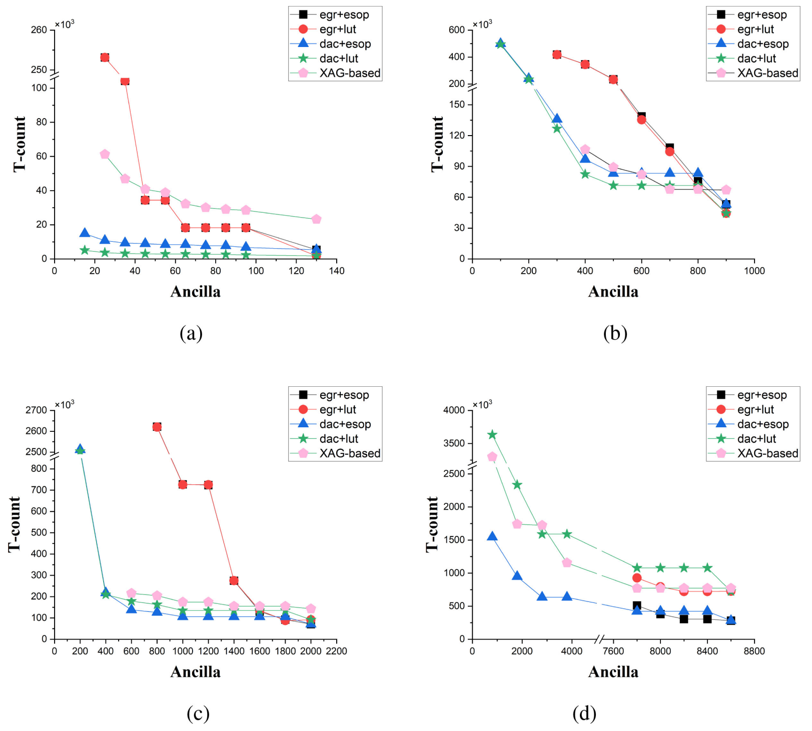

Fig. 6. Qubit-constrained synthesis results. *egr*: eager clean-up pebbling strategy, *dac*: divide-and-conquer pebbling strategy, *esop*: ESOP-based STG synthesis, *lut*: LUT-based STG synthesis. (a) Synthesis results for benchmark Adder. (b) Synthesis results for benchmark Max. (c) Synthesis results for benchmark Sine. (d) Synthesis results for benchmark Square-root.

for the pebbling process is predetermined. In this section, we evaluate the performance of our strategy under qubit-constrained conditions, where the number of available ancilla qubits is specified in advance, and the cut size for the LUT mapping process is adjusted accordingly to meet these constraints.

Fig. 6 presents the T-count of the synthesis results for four benchmarks with varying numbers of ancilla qubits. The Adder benchmark is selected to represent a relatively small case, while Max and Sine are chosen as medium-sized examples, and Square-root represents a relatively large case. The cut sizes for the LUT mapping process are determined through the incremental approach outlined in Section V-A, with minimum and maximum cut sizes set at 4 and 15, respectively.

Two different STG synthesis methods are employed in these experiments. The ESOP-based synthesis, denoted as *esop*, is the same method used in the previous section. The LUT-based synthesis, denoted as *lut*, maps each STG with no more than 4 control qubits into precomputed near-optimal Clifford+T circuits by referencing a database [9]. For STGs with more than four control qubits, the synthesis reverts to the ESOP-based method. Additionally, the results of the XAG-based approach are provided for comparison. Although the XAG-based approach does not include the LUT-mapping stage, the required number of qubits can still be adjusted using the divide-and-conquer strategy. The best-fit mapping technique is not applied in this section and will be explored in the next section.

The results across all benchmarks and combinations of pebbling strategies and STG synthesis methods reveal a consistent pattern: the T-count of the final circuit sharply increases as qubit limitations tighten, following a period of relatively stable performance. However, when employing the divide-and-conquer strategy, this stable period is notably extended. This phenomenon is attributed to our strategy's flexibility in trading off qubit usage for circuit gate count, as discussed in Section IV-B. By leveraging additional pebbling steps, we can delay the need to increase the cut size for a longer duration, allowing the cut sizes to remain minimal until our strategy reaches its limit for conserving qubits.

The divide-and-conquer strategy excels in reducing qubit usage compared to the eager clean-up strategy, enabling valid results under much tighter constraints on the number of ancilla qubits across all cases. In situations where both strategies are applicable, the T-count of results obtained with the divide-and-conquer strategy is generally lower than that of the eager clean-up strategy, particularly under tight constraints. This is because the divide-and-conquer strategy can maintain a smaller cut size than the eager clean-up strategy under the same qubit limits, albeit at the cost of additional pebbling steps. In most cases, the impact of these increased steps on the final circuit gate count does not match the growth of cut sizes. As constraints loosen, the advantages of the divide-and-conquer strategy diminish, and its results may occasionally be worse than those of the eager clean-up strategy. With sufficient ancilla qubits, both strategies effectively operate the same way, resulting in identical outcomes.

The LUT-based STG synthesis method functions identically to the ESOP-based method for STGs with more than four control qubits, differing mainly by directly mapping smaller STGs

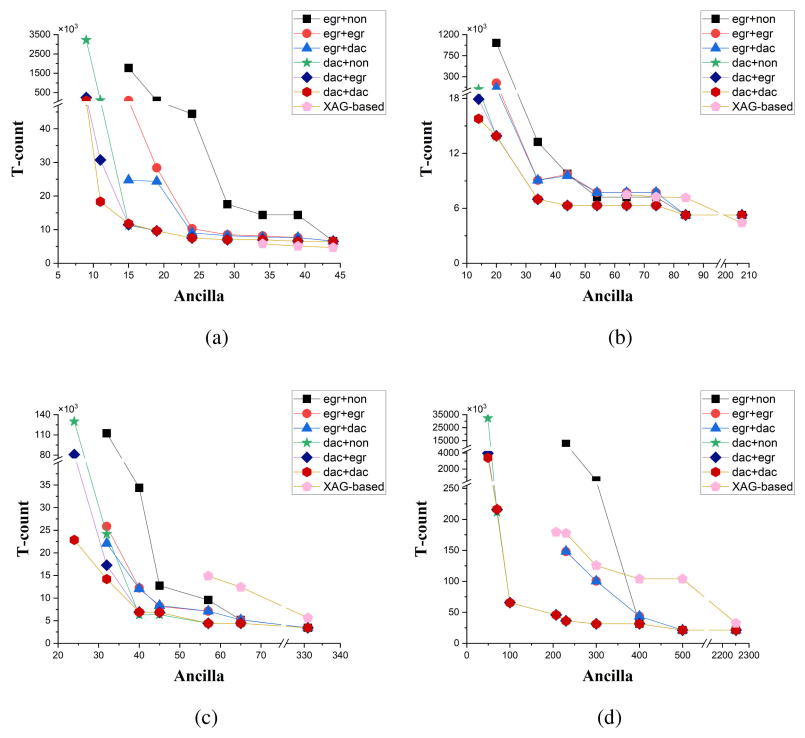

Fig. 7. Qubit-constrained synthesis results with best-fit mapping. *egr*: eager clean-up strategy, *dac*: divide-and-conquer strategy, *non*: no extra LUT mapping. (a) c432, PI: 36, PO: 7, nodes: 246. (b) c880, PI: 60, PO: 26, nodes: 388. (c) c1908, PI: 33, PO: 25, nodes: 448. (d) c6288, PI: 32, PO: 32, nodes: 2370.

into near-optimal circuits. At first glance, this might seem like an improvement over the ESOP-based method, as the exorcism algorithm may struggle to consistently achieve near-optimality for every small STG. However, the results in Fig. 6(c) and (d) suggest otherwise. In many cases, the ESOP-based method actually produces lower T-counts than the LUT-based approach. This is likely because the near-optimal circuits are often computed with few or no ancilla qubits, leaving room for further optimization if more ancilla qubits are available (hence "near-optimal" rather than "optimal"). As a result, the ESOP-based method can achieve better results by utilizing ancilla qubits in decomposing multicontrolled Toffoli gates. The fact that the ESOP-based method outperforms the LUT-based method in a significant number of experiments underscores the effectiveness of combining exorcism with ancilla-aided Toffoli decomposition.

The results of the XAG-based approach follow a similar trend to the other implementations. However, since it typically requires more ancilla qubits to execute (see Section VI-B), the range of its results on the x-axis is narrower compared to the LHRS algorithm when combined with the divide-and-conquer strategy. For instance, on the benchmark Max, the XAG-based approach requires more than 300 ancilla qubits, whereas the LHRS algorithm, when combined with the divide-and-conquer strategy, is able to operate with fewer than 100 ancilla qubits. Additionally, the XAG-based approach demands significantly more ancilla qubits to achieve a sufficiently low T-count. Under the same relatively low qubit constraint, it generally produces a higher T-count in the final circuit than the LHRS algorithm.

D. Optimize STG Sizes With Best-Fit Mapping

The results from the previous section demonstrate that the divide-and-conquer strategy effectively reduces circuit gate count under qubit constraints by enabling lower cut sizes during the LUT mapping process. Its advantage over the eager clean-up strategy is particularly pronounced in tasks with tight ancilla constraints. In this section, we combine our strategy with the best-fit mapping technique (see Section V-B) to show how this combination further reduces the size of STGs (measured by the number of control qubits) and evaluate its impact on the final circuit gate count.

The results of six different implementations are presented in Fig. 7. Each line in Fig. 7 represents a specific combination of pebbling strategies applied in the two rounds of LUT mapping. Here, *egr* signifies the eager clean-up strategy, *dac* represents the divide-and-conquer strategy, and *non* indicates that no second round of LUT mapping (and hence no best-fit mapping) was applied. The combinations "egr+non" and "egr+egr" correspond to the strategies used in the original LHRS algorithm and the best-fit mapping strategy, respectively. The results of the XAG-based approach are provided for comparison. For simplicity, all STGs in this experiment are synthesized using the ESOP-based synthesis method.

We notice in our experiments that on large benchmarks, the T-count of the final circuit is relatively insensitive to small variations in the number of available ancilla qubits, with a significant gap typically existing between the minimum ancilla requirement of the eager clean-up strategy and the divide-and-conquer strategy. Therefore, it is challenging to capture precise changes caused by small adjustments in the available ancilla resources while still maintaining a comprehensive analysis in such cases. To address this, we shift to the ISCAS85 benchmark, where smaller cases allow for clearer tracking of changes in the T-count incurred by incremental adjustments in the number of available ancilla qubits. Fig. 7(a), (b), and (c) show the number of T gates in the synthesized circuits for benchmarks c432, c880, and c1908, respectively, with the number of primary inputs (PI), primary outputs (PO), and the number of nodes before LUT mapping listed below each figure. For comparison, the results of the relatively large benchmark c6288 are shown in Fig. 7(d).

The results across all four benchmarks exhibit the same pattern as observed in the previous section: the number of T gates decreases as the number of available ancilla qubits increases. The divide-and-conquer strategy, when applied during the first round of LUT mapping, consistently outperforms the eager clean-up strategy until sufficient ancilla resources are available, at which point both strategies converge to identical results. However, reducing the incremental step in the number of available ancilla qubits uncovers certain exceptions. For instance, in benchmark c880, using 44 ancilla qubits results in a higher T-count than using 34 ancilla qubits when applying the eager clean-up strategy in the first and second round of LUT mapping. This happens because the advantage of reducing cut sizes does not always outweigh the additional computational steps required. Moreover, decomposing the logic network into smaller components can sometimes miss optimization opportunities. This indicates a potential direction for further research into better synchronization between the LUT mapping process and STG synthesis. Nevertheless, minimizing cut sizes remains an effective approach for reducing circuit gate count at this stage.

Furthermore, compared to cases where only one round of LUT mapping is applied, those incorporating an additional round generally achieve better results. Circuits that employ the divide-and-conquer strategy in the second round tend to produce lower T-counts in most experiments. The difference is most pronounced under tight qubit constraints and diminishes when ample ancilla resources are available, as the first round of LUT mapping already achieves the desired effect. The distributions of STG sizes for all four benchmarks with the minimum number of ancilla qubits are depicted in Fig. 8. In each distribution, it can be seen that using the divide-and-conquer strategy in the second round of LUT mapping results in a significantly higher number of small-sized STGs (particularly those with no more than four control qubits) and further reduces the large-sized STGs, compared to the eager clean-up strategy. The greater this difference, the more noticeable the performance improvement in Fig. 7.

There are still exceptions where applying the best-fit mapping technique increases the circuit's T-count, which occurs for the same reason mentioned in the previous paragraph. This can be addressed by applying the *pick-best* method [28], where each STG undergoes a second round of LUT mapping only if it reduces the total T-count. Since this approach operates on each individual STG rather than the entire circuit, it can lead to a lower T-count even in cases where the best-fit mapping technique already has a positive effect.

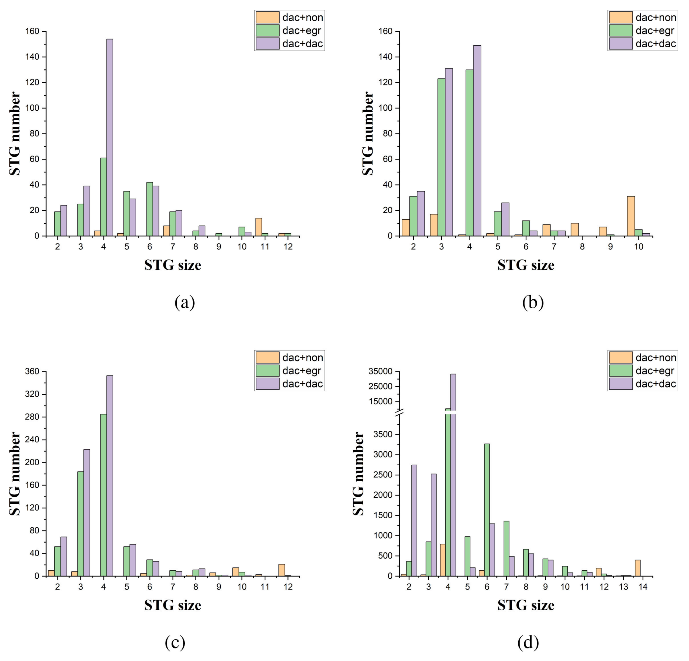

Fig. 8. Distribution of STG sizes with minimum ancilla resource. (a) Benchmark c432 with nine ancilla qubits. (b) Benchmark c880 with 14 ancilla qubits. (c) Benchmark c1908 with 24 ancilla qubits. (d) Benchmark c6288 with 49 ancilla qubits.

Similar to the experiments in Section VI-C, under the same qubit constraint, the XAG-based approach generally yields a higher T-count than the LHRS algorithm, except for a few cases. When provided with sufficient ancilla qubits, it outperforms the LHRS algorithm on benchmarks c432 and c880. However, on benchmarks c1908 and c6288, it is outperformed by the combination of the LHRS algorithm and the best-fit mapping technique. Furthermore, the number of additional ancilla qubits required for the XAG-based approach to achieve its optimal result is significantly higher than that of the LHRS algorithm. These observations suggest that the advantage of the XAG-based approach over the LHRS algorithm may not be consistently evident. A hybrid approach combining the two methods could be a promising direction for future research.

E. Intergration Into Qiskit

We have implemented our method as a potential synthesizer for Qiskit's PhaseOracle and BooleanExpression classes. In this section, we evaluate the performance of our proposed method in comparison to Qiskit's default synthesis approach, denoted as *pkrm_synth*. Our experiments revealed that the *pkrm_synth* implementation encounters errors when handling cases with more than ten variables. Due to this uncertainty, we opted to focus on smaller benchmarks. Since suitable benchmarks of this scale are difficult to find, we chose to test both methods on random CNF cases generated by CNFgen [34]. Fig. 9 illustrates the average CNOT count and CNOT depth for the synthesis results of *pkrm_synth*, along with the outcomes of our method using 0 and 5 ancilla qubits. The STG synthesis in these experiments is accomplished using the ESOP-based synthesis, and multicontrolled

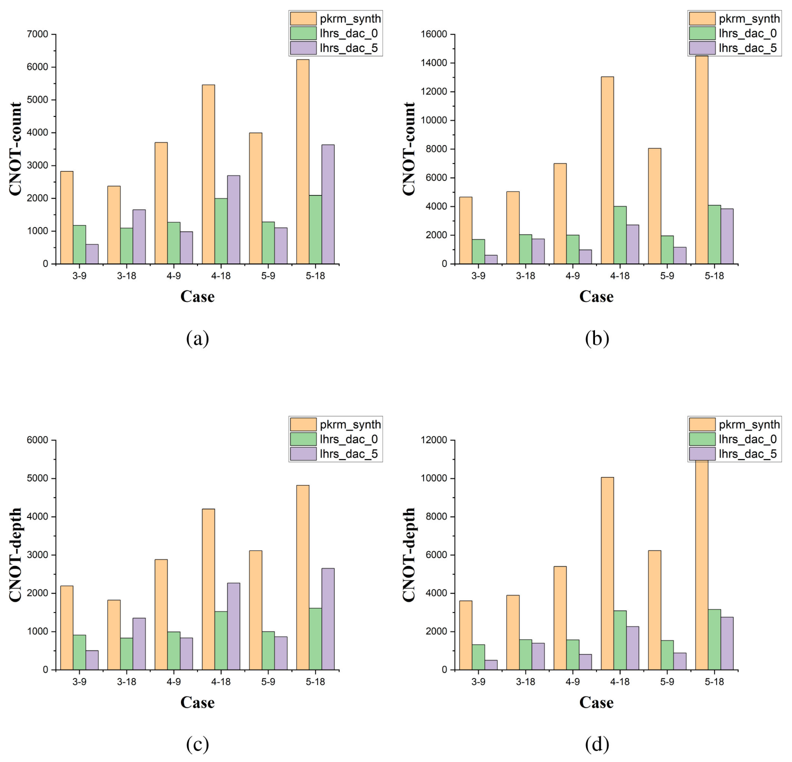

Fig. 9. Comparison of our method to Qiskit's default method. *pkrm_synth*: Qiskit's default method, *lhrs_dac_0*: Our method with 0 ancilla qubit, *lhrs_dac_5*: Our method with 5 ancilla qubits. The cases are identified by the first digit representing the number of literals in each CNF clause, while the second digit indicates the number of clauses. (a) Average CNOT count for random CNF cases with 8 variables. (b) Average CNOT count for random CNF cases with 9 variables. (c) Average CNOT depth for random CNF cases with 8 variables. (d) Average CNOT depth for random CNF cases with 9 variables.

Toffoli networks are decomposed into basic gates using the *decompose* method in Qiskit.

The proposed method demonstrates a clear advantage over the *pkrm_synth* method across all test cases, particularly when aided by five ancilla qubits. However, in some instances, the inclusion of five ancilla qubits results in a higher cost in the final circuit. This occurs because the ancilla qubits enable a smaller cut size during LUT mapping, which does not always translate to a lower final circuit cost, as previously discussed. Nonetheless, our method outperforms *pkrm_synth* even under these circumstances.

VII. Conclusion

We propose a novel pebbling strategy based on divide-and-conquer for the reversible pebble game, which can seamlessly integrate into the LHRS algorithm and the XAG-based approach. Experiments on the EPFL benchmarks demonstrate that our algorithm can effectively handle large-scale cases with notable efficiency, achieving up to a 90% reduction in qubits while incurring an average increase of less than ten times the original steps. Moreover, our strategy is adept at addressing qubit-constrained tasks by minimizing the cut sizes for LUT mapping, thereby resulting in a substantial reduction in circuit gate count.

References

[1] L. Li, F. Voichick, K. Hietala, Y. Peng, X. Wu, and M. Hicks, "Verified compilation of quantum oracles," in *Proc. ACM Program. Lang.*, vol. 6, 2022, pp. 589–615.

[2] A. Ambainis, "Understanding quantum algorithms via query complexity," in *Proc. Int. Congr. Math.*, 2018, pp. 3265–3285.

[3] J. M. Henderson, E. R. Henderson, A. Sinha, M. A. Thornton, and D. M. Miller, "Automated quantum oracle synthesis with a minimal number of Qubits," 2023, *arXiv:2304.03829*.

[4] B. Schmitt, F. Mozafari, G. Meuli, H. Riener, and G. De Micheli, "From boolean functions to quantum circuits: A scalable quantum compilation flow in C++," in *Proc. IEEE Design Autom. Test Europe Conf. Exhibit. (DATE)*, 2021, pp. 1044–1049.

[5] A. Parent, M. Roetteler, and K. M. Svore, "REVS: A tool for space-optimized reversible circuit synthesis," in *Proc. 9th Int. Conf. Reversible Comput. (RC)*, 2017, pp. 90–101.

[6] *Qiskit: An Open-Source Framework for Quantum Computing*, Qiskit, New York, NY, USA, 2023.

[7] K. Svore et al., "Q# enabling scalable quantum computing and development with a high-level DSL," in *Proc. Real World Domain Specific Lang. Workshop*, 2018, pp. 1–10.

[8] K. Fazel, M. A. Thornton, and J. E. Rice, "ESOP-based Toffoli gate cascade generation," in *Proc. IEEE Pac. Rim Conf. Commun. Comput. Signal Process.*, 2007, pp. 206–209.

[9] M. Soeken, M. Roetteler, N. Wiebe, and G. De Micheli, "LUT-based hierarchical reversible logic synthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 9, pp. 1675–1688, Sep. 2019.

[10] C. H. Bennett, "Time/space trade-offs for reversible computation," *SIAM J. Comput.*, vol. 18, no. 4, pp. 766–776, 1989.

[11] B. Komarath, J. Sarma, and S. Sawlani, "Pebbling meets coloring: Reversible pebble game on trees," *J. Comput. Syst. Sci.*, vol. 91, pp. 33–41, Feb. 2018.

[12] S. M. Chan, "Just a pebble game," in *Proc. IEEE Conf. Comput. Complexity*, 2013, pp. 133–143.

[13] G. Meuli, M. Soeken, M. Roetteler, N. Bjorner, and G. De Micheli, "Reversible pebbling game for quantum memory management," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, 2019, pp. 288–291.

[14] D. Bhattacharjee, M. Soeken, S. Dutta, A. Chattopadhyay, and G. De Micheli, "Reversible pebble games for reducing qubits in hierarchical quantum circuit synthesis," in *Proc. IEEE 49th Int. Symp. Multiple Valued Logic (ISMVL)*, 2019, pp. 102–107.

[15] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *Proc. 24th Int. Workshop Logic Synth. (IWLS)*, 2015, pp. 1–8.
[16] G. Meuli, M. Soeken, and G. De Micheli, "XoR-and-inverter graphs for quantum compilation," *NPJ Quant. Inf.*, vol. 8, no. 1, p. 7, 2022.
[17] A. Barenco et al., "Elementary gates for quantum computation," *Phys. Rev. A Atomic Mol. Opt. Phys.*, vol. 52, no. 5, p. 3457, 1995.
[18] J. M. Baker, C. Duckering, A. Hoover, and F. T. Chong, "Decomposing quantum generalized Toffoli with an arbitrary number of Ancilla," 2019, *arXiv:1904.01671*.
[19] T. Sasao, "An exact minimization of AND-EXOR expressions using BDD's," in *Proc. IFIP WG*, 1993, pp. 1–8.
[20] R. Drechsler, "Pseudo-Kronecker expressions for symmetric functions," *IEEE Trans. Comput.*, vol. 48, no. 9, pp. 987–990, Sep. 1999.
[21] A. Mishchenko and M. Perkowski. "Fast heuristic minimization of exclusive-sums-of-products." 2001. [Online]. Available: https://people.eecs.berkeley.edu/ alanmi/publications/2001/rm01_heu.pdf
[22] G. Meuli, B. Schmitt, R. Ehlers, H. Riener, and G. De Micheli, "Evaluating ESOP optimization methods in quantum compilation flows," in *Proc. 11th Int. Conf. Reversible Comput. (RC)*, 2019, pp. 191–206.
[23] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2007, pp. 354–361.
[24] S. Ray, A. Mishchenko, N. Een, R. Brayton, S. Jang, and C. Chen, "Mapping into LUT structures," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, 2012, pp. 1579–1584.
[25] G. Meuli, M. Soeken, M. Roetteler, and G. De Micheli, "ROS: Resource-constrained oracle synthesis for quantum computers," 2020, *arXiv:2005.00211*.
[26] C. H. Bennett, "Logical reversibility of computation," *IBM J. Res. Develop.*, vol. 17, no. 6, pp. 525–532, 1973.
[27] E. Knill, "An analysis of Bennett's pebble game," 1995, *arXiv:math/9508218*.
[28] G. Meuli, M. Soeken, M. Roetteler, N. Wiebe, and G. De Micheli, "A best-fit mapping algorithm to facilitate ESOP-decomposition in clifford+T quantum network synthesis," in *Proc. 23rd Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2018, pp. 664–669.
[29] H. Y. Althoby, M. D. Biha, and A. Sesboüé, "Exact and heuristic methods for the vertex separator problem," *Comput. Ind. Eng.*, vol. 139, Jan. 2020, Art. no. 106135.
[30] Y. Ding et al., "Square: Strategic quantum Ancilla reuse for modular quantum programs via cost-effective uncomputation," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Architect. (ISCA)*, 2020, pp. 570–583.
[31] M. Soeken et al., "The EPFL logic synthesis libraries," 2018, *arXiv:1805.05121*.
[32] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proc. Int. Conf. Tools Algorithms Construction Anal. Syst.*, 2008, pp. 337–340.
[33] E. Kashefi, A. Kent, V. Vedral, and K. Banaszek, "Comparison of quantum oracles," *Phys. Rev. A Atomic Mol. Opt. Phys.*, vol. 65, no. 5, 2002, Art. no. 50304.
[34] M. Lauria, J. Elffers, J. Nordström, and M. Vinyals, "CNFgen: A generator of crafted benchmarks," in *Proc. 20th Int. Conf. Theory Appl. Satisfiability Testing (SAT)*, 2017, pp. 464–473.

Kezhen Zhang received the B.Sc. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2021. He is currently pursuing the Ph.D. degree from the Institute of Software, Chinese Academy of Sciences, Beijing.

His research interests include quantum computing and quantum circuit synthesis.

Riling Li received the B.Sc. and Ph.D. degrees from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2016 and 2021, respectively.

He is currently a Postdoctoral Researcher with the Institute of Software, Chinese Academy of Sciences, Beijing. His research interests are quantum programming and quantum circuits.

Mingsheng Ying received the bachelor's degree from Fuzhou Teachers College, Fuzhou, Jiangxi, China, in 1981.

He is currently a Distinguished Professor with the Centre for Quantum Software and Information, University of Technology Sydney, Sydney, NSW, Australia. He was a Research Professor and the Deputy Director for research with the Institute of Software, Chinese Academy of Sciences, Beijing, China, and the Cheung Kong Professor with the Department of Computer Science and Technology, Tsinghua University, Beijing. He has authored the books *Model Checking Quantum Systems: Principles and Algorithms* (Cambridge University Press, 2021), *Foundations of Quantum Programming* (Morgan Kaufmann, first edition 2016; second edition 2024), and *Topology in Process Calculus: Approximate Correctness and Infinite Evolution of Concurrent Programs* (Springer-Verlag, 2001). His research interests include quantum computing, programming language theory, and logics in artificial intelligence.

Mr. Ying currently serves as the (Co-)Editor-In-Chief of for the *ACM Transactions on Quantum Computing*.