A New Perspective on Constructing Resource-Efficient Data-Lookup Quantum Oracles

Mingfei Yu
Integrated Systems Laboratory, EPFL
Lausanne, Switzerland
mingfei.yu@epfl.ch

Mathias Soeken
Microsoft
Zurich, Switzerland
mathias.soeken@microsoft.com

Giovanni De Micheli
Integrated Systems Laboratory, EPFL
Lausanne, Switzerland
giovanni.demicheli@epfl.ch

***Abstract*—Quantum oracles are essential components in quantum algorithms, enabling the efficient evaluation of Boolean functions. This paper presents a novel approach to constructing resource-efficient data-lookup quantum oracles for *fault-tolerant quantum computing* (FTQC) systems. By leveraging the *algebraic normal form* (ANF) of Boolean functions, rather than the truth tables used in existing approaches, we achieve a provable $\mathcal{O}(n)$ reduction in the minimum Toffoli gate count — a key cost metric in FTQC — for constructing an n-variable data-lookup oracle. Additionally, exploiting ANF product terms allows for the exploration of space-time trade-offs, enabling the design of data-lookup oracles with distinct resource requirements that can adapt to varying system constraints — flexibility that current methods do not offer. Based on the target quantum system's requirements, our approach offers the ability to tailor oracle designs to minimize either Toffoli count and depth, or ancillary qubit usage.**

***Index Terms*—Quantum oracle; Logic synthesis; Reversible computing; Fault-tolerant quantum computing**

I. Introduction

To run a quantum algorithm on a quantum computer, the program has to be compiled into a *quantum circuit*, where each gate is an operation native to the quantum hardware system. Specifically, a quantum circuit is also referred to as a *quantum oracle* when it implements a Boolean function.

Quantum oracles are a fundamental component in many quantum algorithms. As an example, the *Shor's algorithm* is a well-known quantum algorithm where an oracle is utilized [1]. The algorithm is used for factoring large integers and computing discrete logarithms. A key to Shor's algorithm is *quantum phase estimation* that identifies the period of a function. This step relies on a modular exponentiation function, in the construction of which the oracle plays a crucial role. Quantum oracles also find uses in various *quantum machine learning* algorithms, such as *quantum nearest neighbor* algorithms [2]. In this scope, oracles can be used to encode datasets or compute distance functions between data points. Due to the uses of quantum oracles across various application scenarios, their design and optimization are crucial for realizing the full potential of quantum computing.

Intensive efforts have been put into exploring how to compile a Boolean function into a quantum circuit. One line of research focuses on using logic representations as intermediate forms for constructing quantum oracles, like *XOR-AND-inverter graphs* (XAG) [3]. The rationale behind this strategy lies in the relationship between the properties of these intermediate logic representations and the resources required for the resulting quantum oracles. For example, the number of AND nodes in an XAG directly correlates with the T count of the compiled oracle [4], a critical cost metric in *fault-tolerant quantum computing* (FTQC) systems. By exploiting these relationships, a resource-efficient oracle can be generated by first optimally synthesizing the logic representation of the target Boolean function and then deterministically compiling this intermediate representation into a quantum circuit. Consequently, the structures of quantum oracles of different functions derived through this process can vary significantly.

In contrast, data-lookup oracles, also known as *quantum lookup tables* [5]–[7], represent a distinctive approach to quantum oracle construction. They are characterized by their generic design that does not depend on the specific Boolean function being targeted. Although this approach cannot leverage the properties of the target Boolean function to minimize quantum resources, it is often preferred in practice for several practical reasons. First, a generic construction ensures a uniform layout, a crucial feature for practical quantum applications. Second, data-lookup oracles are particularly well-suited for application scenarios where target functions require frequent reconfiguration. For example, many quantum arithmetic algorithms are designed to accommodate different levels of approximation [8]. Data-lookup oracles inherently provide this flexibility, while alternative approaches would necessitate a new oracle design for each specific parameter configuration.

In addition to the truth table, the *algebraic normal form* (ANF) is another canonical representation of Boolean functions. In this work, we propose a novel method for constructing data-lookup quantum oracles by utilizing the ANF of Boolean functions. Unlike conventional approaches that rely on iterating through truth table entries, our method focuses on the product terms present in the ANF. This new perspective reduces the minimum required Toffoli gates by $\mathcal{O}(n)$ and enables a reduction in Toffoli depth, as many product terms can be computed in parallel without data contention.

Additionally, we formulate the oracle construction problem as a two-stage process, introducing a space-time trade-off that provides flexibility in optimizing various quantum resource constraints. Specifically, our approach decouples the construction into (1) synthesizing an AND forest to compute the

product terms, and (2) applying a modified reversible pebbling game to optimize the resulting quantum circuit in terms of Toffoli count and depth, and ancillary qubit usage.

The remainder of the paper is organized as follows: Section II covers the necessary background. Section III outlines our proposed oracle construction method, with a detailed SAT formulation presented in Section IV. Section V presents alternative constraints that improve the scalability of the SAT formulation while maintaining a high-quality solution space. Section VI contains a comprehensive evaluation of our oracle constructions, and we conclude with future directions in Section VII.

II. Background

A. Quantum Oracles

To run a quantum algorithm on a quantum computer, the algorithm must be decomposed into a specific set of operations supported by the target system. The *Clifford+T* gate library is commonly used in studies of FTQC systems, with designs that minimize T count and T depth being preferred, as T gates are significantly more resource-intensive than Clifford gates [9].

In this work, we consider a gate library consisting of {Toffoli (CCX), Controlled-NOT (CX), NOT (X)}. Our choice is motivated by two factors when compiling this set into the Clifford+T gate library: (i) the Toffoli gate is the only gate in this set that requires T gates for its implementation, and (ii) there is a well-established linear relationship between Toffoli count/depth and T count/depth [10]. By adopting this higher-level gate library, we simplify the analysis while still providing an accurate cost estimation for the constructed quantum oracles. This approach is consistent with the gate library selection in other relevant research, such as [5].

When implementing a Boolean function, a quantum circuit is specifically called a *quantum oracle*. Given a Boolean function $f(x_1, \ldots, x_n) = (y_1, \ldots, y_m) : \mathbb{B}^n \to \mathbb{B}^m$, with $x = x_1 \ldots x_n$, $y = y_1 \ldots y_m$ and bitwise XOR operations denoted as '$\oplus$', the corresponding quantum oracle can be described as $O_f : |x\rangle |y\rangle |0\rangle^l \mapsto |x\rangle |y \oplus f(x)\rangle |0\rangle^l$. Besides n qubits and m qubits to respectively store the inputs and outputs, oracle O_f requires l ancillary qubits to hold intermediate results, which shall be uncomputed to the initial $|0\rangle$ state at the end of the computation, to guarantee an accurate measurement.

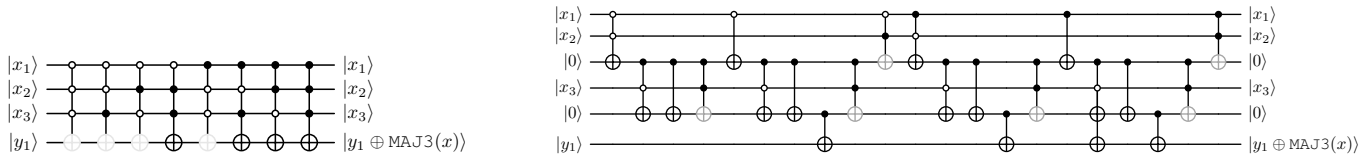
1) Data-lookup Quantum Oracles: Data-lookup quantum oracles, sometimes referred to as *quantum lookup tables* [7], are used to retrieve function values by querying pre-defined entries based on a given input. Existing state-of-the-art constructions typically rely on iterating through all entries of a truth table. In this work, we assume the target function f is fully specified for n variables, meaning that for any n-bit input, the function's output is explicitly defined.

2) Existing Works: State-of-the-art data-lookup quantum oracle constructions for an n-variable function are typically based on iterating over the 2^n entries in the truth table representation of the target function. The index state controls the state transformation, a concept commonly referred to as a *select network*. This approach is illustrated in Fig. 1a, using the three-input majority function, `MAJ3`, as an example. The eight minterms are processed sequentially: If a minterm is in the onset of the target function (i.e., the function evaluates to true for the corresponding input pattern), it contributes to the output qubit. Otherwise, it does not (faded in Fig. 1a).

Building on this idea, the *sawtooth-circuit*-based construction offers a T-count-optimized version of the naïve approach [5], as shown in Fig. 1b. In this construction, the multi-controlled NOT gates from the naïve design are decomposed into Toffoli gates, and a set of transformation rules is applied to replace adjacent Toffoli gates with CNOT and NOT gates, reducing the overall Toffoli count. The sawtooth-circuit-based construction is deterministic and has a well-defined quantum resource requirement: For an n-variable function, it results in a Toffoli count and depth of 2^n-2, using $n-1$ ancillary qubits.

Another widely recognized construction for data-lookup quantum oracles is the *select-swap network* [6]. In addition to the select network discussed earlier, a swap network is employed to provide a flexible trade-off between Toffoli count and ancillary qubit usage. Since the construction proposed in this paper serves as an alternative to the select network and is compatible with the swap network, our focus remains on the select network.

Other efforts to optimize data-lookup oracle designs, such as in [11], employ techniques like *pre-decoding* to reduce reliance on multi-controlled Toffoli gates. However, these approaches are generally tailored for *noisy intermediate-scale quantum* (NISQ) systems and do not reduce the Toffoli count — the key cost metric in FTQC systems, which is the focus of this work. As such, these approaches are orthogonal to



(a) (b)

Fig. 1: Realization of `MAJ3` using data-lookup quantum oracle constructions based on iterating the 2^3 entries of the truth table representation of the target function: (a) The naïve starting-point construction, where non-contributing minterms are faded. (b) The sawtooth-circuit-based construction, with uncomputation indicated in gray.

our contribution. To the best of our knowledge, the sawtooth-circuit-based construction remains the state-of-the-art [12], making it a suitable benchmark for comparison in this study.

B. Directed Acyclic Graph

A *directed acyclic graph* (DAG) $G = (V, E)$ consists of a set of nodes V and a set of directed edges E. Each edge in E represents a connection from node n_i to node n_o, where $n_i, n_o \in V$. We refer to n_i as the *fan-in* of n_o and n_o as the *fan-out* of n_i. A DAG is also characterized by its *primary inputs* (PIs) and *primary outputs* (POs). Each node in V is either a PI or an internal node. Each PO can be either a PI or an internal node. While this is not a strict requirement in general, for clarity in this work, we consider nodes without a fan-out as POs.

C. Reversible Pebbling Game

A reversible pebbling game is a computational model that explores the space-time trade-off in quantum computation [13]. The game is typically played on a DAG representing a computation. In this context, each node in the DAG corresponds to a computational step, and pebbles represent the availability of computed results stored on qubits.

Initially, no node is pebbled, and the goal is to pebble the POs while minimizing the number of pebbles used or computational steps taken. The rules of the game dictate that a pebble can be placed on or removed from a node only if all its fan-in nodes are already pebbled. Moreover, a pebble can only be removed from a node if all its fan-ins remain pebbled. This adds complexity to managing intermediate computational results since it mirrors the challenge of uncomputing temporary data in quantum computations without excessive usage of ancillary qubits.

Solving a reversible pebbling game reveals key trade-offs between the ancillary qubit count (pebbles) and the depth (steps) of the computation, commonly known as the space-time trade-off. Different pebble strategies applied to a DAG correspond to quantum algorithm implementations that require varying amounts of quantum resources. Existing research has introduced a *Boolean satisfiability* (SAT) encoding for solving reversible pebbling games on a given DAG efficiently [14].

D. Algebraic Normal Form

The *algebraic normal form* (ANF) is an alternative canonical representation of Boolean functions. It expresses the function as an exclusive sum of product terms involving the variables, where each product term corresponds to a conjunction of Boolean literals. Formulas written in ANF are also known as *positive-polarity Reed-Müller expressions* (PPRM), as any negation on input variables is not allowed. Specifically, the ANF of an n-variable singular-output Boolean function $f(x) : \mathbb{B}^n \to \mathbb{B}$ is given by:

$$f(x) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus \cdots \oplus a_{2^n-1} x_1 x_2 \ldots x_n, \quad (1)$$

where $a_0, \ldots, a_{2^n-1}$ are binary coefficients and $x_1, \ldots, x_n$ are Boolean variables, with '$\oplus$' denoting the Boolean XOR operation. Each coefficient determines whether the corresponding product term appears in the function. For clarity, we refer to the terms that contain more than one input variable as the *product terms*. The ANF of an n-variable functions has $\sum_{i=2}^{n} \binom{n}{i} = 2^n - n - 1$ product terms.

Any Boolean function can be uniquely represented in ANF using only Boolean AND and XOR operations. Following the *Davio expansion*, a Boolean function can be decomposed as

$$f(x) = f_{x_i=0}(x) \oplus x_i (f_{x_i=0}(x) \oplus f_{x_i=1}(x)), \quad (2)$$

where $f_{x_i=0}(x)$ and $f_{x_i=1}(x)$ denote the $(n-1)$-variable Boolean functions obtained by assigning the i-th variable of function f to 0 and 1, respectively. These are referred to as the *negative* and *positive co-factors* of f concerning x_i. By recursively applying Eq. (2) to the truth table representation of a Boolean function, its ANF representation can be derived deterministically and efficiently.

III. Constructing Better Data-lookup Oracles

In this section, we present an overview of our approach to constructing better data-lookup oracles and highlight the technical challenges to address, to facilitate this idea.

A. An $\mathcal{O}(n)$ Toffoli Count Reduction via ANF

Our approach to constructing an optimized n-variable data-lookup oracle involves enumerating the product terms in the ANF of an n-variable Boolean function. As can be learned from Eq. (1), in the ANF of a function, there are $2^n - n - 1$ potential terms that contain more than one input variable, the computation of each of which relies on applying Boolean AND operation to some product terms containing fewer variables.

Theorem 1. *Computing all $2^n - n - 1$ product terms of an n-variable Boolean function requires $2^n - n - 1$ two-input Boolean AND operations.*

Proof. The lower bound is straightforward: each AND operation computes only one product term, so at least $2^n - n - 1$ operations are required.

For the upper bound, assume the computation proceeds level-by-level. Specifically, product terms involving d variables ($2 \leq d \leq n$) are computed only after all terms involving $d-1$ variables are computed. Each such product term is obtained by applying one AND operation between an $(d-1)$-variable product term and an additional variable. Thus, every product term requires exactly one AND operation, resulting in a total of $2^n - n - 1$ two-input AND operations. A similar upper bound analysis is provided in [15]. ■

This approach fundamentally differs from enumerating the 2^n minterms used in the truth table representation, as seen in the state-of-the-art construction [5]. Since each two-input AND operation can be implemented with a single Toffoli gate, our novel method for constructing data-lookup oracles deterministically reduces the number of Toffoli gates by n, leading to an $\mathcal{O}(n)$ reduction in Toffoli count.

In Fig. 2, we demonstrate the data-lookup oracle constructed following the proposed idea, for the specific case where the

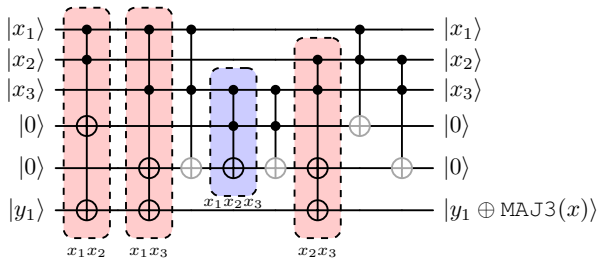


Fig. 2: Realization of MAJ3 using the proposed data-lookup quantum oracle construction. Product terms contributing to the target function are highlighted in red, while non-contributing terms are marked in blue, with the corresponding computed term listed below each. Uncomputation of product terms is indicated in gray.

target function is also MAJ3, as a comparison to the sawtooth-based construction in Fig. 1b. The ANF of MAJ3 is given by:

$$\texttt{MAJ3}(x) = x_1x_2 \oplus x_1x_3 \oplus x_2x_3. \tag{3}$$

While all four product terms are computed (marked either in red or blue in Fig. 2), only the three terms in Eq. (3) contribute to the output via CNOT gates (highlighted in red). It is important to note that the uncomputation of each Toffoli gate (marked in gray) does not require any T gates [10]. The resource requirement of the construction in Fig. 2 is four Toffoli gates, which are highlighted, with a depth of four since they are executed sequentially, and two ancillary qubits. Compared to the SoTA design in Fig. 1b, our construction reduces the Toffoli count and depth by one-third (4 vs. 6) with the same amount of ancillary qubits, making it a significantly more resource-efficient solution.

B. Formulating the Oracle Construction Problem

While the use of ANF suggests a foundation for constructing Toffoli-count-efficient data-lookup oracles, it does not, on its own, yield a complete design. Nor is it guaranteed that the minimum requirement on Toffoli gates can be consistently achieved by any oracle construction based on ANF. To realize a valid construction, two key decisions must be made regarding the computation of the $2^n - n - 1$ product terms: (i) Determining how each product term is computed. For instance, the product term $x_1x_2x_3$ can be derived from x_1x_3 and x_2, as shown in Fig. 2, or alternatively from x_1x_2 and x_3, leading to different circuit designs. (ii) Defining how this computation is efficiently mapped onto a quantum circuit. In the remainder of this section, we address these two challenges by formulating them as design automation problems.

1) AND Forest: We introduce the concept of an *AND forest* to formalize the structural and functional aspects of computing product terms for an n-variable Boolean function. PIs correspond to the input variables, and POs, or roots of the forest, are nodes that do not have any fan-out. The task is to synthesize an AND forest where each node computes one of the $2^n - n - 1$ product terms required by the function. Structurally, an AND forest is a DAG with n PIs, representing the input variables of the Boolean function. The forest contains $2^n - n - 1$ internal nodes, each with an in-degree of two. Functionally, each node in the forest performs a Boolean AND operation, and collectively, the forest computes the $2^n - n - 1$ product terms of the function's ANF. Thus, a *valid* AND forest satisfying these requirements ensures that the product terms are computed. The first decision-making problem — how each product term is computed — is resolved by synthesizing such a forest.

2) Modified Reversible Pebbling Game: Compiling the computation of the product terms onto a quantum circuit can be viewed as applying a reversible pebbling game to an AND forest. However, our formulation introduces two key modifications compared to the traditional reversible pebbling game on a DAG [14]. First, unlike the standard goal of pebbling all POs in the DAG, in our modified game, all product terms must eventually be uncomputed, as they serve as intermediate results that potentially contribute to the final result. Hence, the last step of a valid pebbling strategy requires that no nodes in the AND forest remain pebbled. Second, to prevent trivial strategies where nodes are never pebbled, we impose the condition that each node in the AND forest must be pebbled at least once during the process. This ensures that all product terms are computed before being uncomputed.

In summary, constructing a data-lookup oracle based on ANF can be divided into two sub-problems: (i) synthesizing an AND forest to compute the product terms, and (ii) applying a modified reversible pebbling game to implement this computation as a quantum circuit. While any solution to these tasks yields a functional oracle, an optimal solution minimizes resource metrics. Therefore, as will be discussed in the following section, it is crucial to define optimality and explore efficient methods for identifying such solutions.

A detailed analysis of the relationship between a reversible pebbling strategy and the resulting Toffoli count is not included, as a Toffoli-count-optimal oracle design — one that achieves the theoretical minimum established in Section III-A — can always be obtained by simply enforcing an additional constraint in the reversible pebbling strategy: each required product term is computed exactly once. Given this straightforward solution, our focus in the following sections shifts to exploring how different structural properties of the synthesized AND forest influence Toffoli depth and ancillary qubit count — two critical metrics in fault-tolerant quantum computing that demand more nuanced optimization.

IV. AND Forest Synthesis

As discussed in Section III-B1, building an oracle based on this approach requires a method to compute these product terms, which translates to synthesizing an AND forest. In this section, we progressively address this challenge. First, we introduce a SAT formulation of the synthesis problem, where each solution corresponds to a valid AND forest that satisfies the requirements. Next, we devise and apply additional constraints to the baseline formulation, guiding the

synthesized AND forest to exhibit certain structural properties. This ensures that the reversible pebbling strategy for the synthesized forest will further minimize quantum resource requirements, either by reducing the number of ancillary qubits or optimizing Toffoli depth.

A. Baseline SAT Formulation

1) Variables: The proposed formulation relies on two types of variables: *function* and *connectivity* variables. A function variable $f_{i,j}$ ($n < i \leq 2^n$ and $1 \leq j \leq n$) determines if input variable x_j appears in the product term at node n_i. A connectivity variable $c_{i,k}$ ($1 \leq k < i$) describes if node n_k (or PI x_k, if $k \leq n$) serves as a fan-in for node n_i.

Only the connectivity variables serve as the variables of the SAT instance. Once the values of the connectivity variables are determined, the function variables are automatically assigned, guided by the first type of constraints to be introduced later.

2) Constraints: Three types of constraints are introduced to ensure that each solution of the SAT instance yields a valid AND forest. We denote conjunction and disjunction as '$\wedge$' and '$\vee$', respectively.

The first type of constraint ensures that the product term realized by each node is consistent with its fan-in connections. Specifically, since each node performs a Boolean AND operation, the product term at node n_i contains the input variable x_j only if at least one of its fan-in nodes realizes a product term that includes x_j. This relationship can be expressed symbolically as:

$$f_{i,j} = \bigvee_{k=1}^{i-1} (c_{i,k} \wedge f_{k,j}). \tag{4}$$

The second type of constraint ensures that each node n_i has exactly two fan-in nodes, which is expressed as:

$$\sum_{k=1}^{i-1} c_{i,k} = 2. \tag{5}$$

This constraint presents a *Boolean cardinality* problem that has been extensively studied, with several encoding strategies proposed in the literature. Given the small cardinality in our case, we adopt *binomial encoding* [16] to represent this constraint efficiently in our implementation.

The final type of constraint ensures that all $2^n - n - 1$ desired product terms are computed. Each product term is represented as an n-bit binary string $b = b_1 \dots b_n$, where each bit is set to 1 if the term contains the corresponding input variable, and 0 otherwise. To guarantee that each product term b is realized by at least one of the $2^n - n - 1$ internal nodes in the synthesized AND forest, the following clause is introduced:

$$\bigvee_{i=n+1}^{2^n} (\bigwedge_{j=1}^{n} f'_{i,j}), \tag{6}$$

where $f'_{i,j} = f_{i,j}$ if $b_j = 1$ and $f'_{i,j} = \neg f_{i,j}$ if $b_j = 0$, with '$\neg$' denoting negation. Ensuring that all desired product terms are implemented by the AND forest also guarantees that no internal nodes in the forest compute the same product term.

Any solution to a SAT instance incorporating the constraints outlined above yields a valid AND forest, as defined in Section III-B1. However, this alone does not fully determine the oracle design or the critical resource metrics. If we ensure that each node is pebbled exactly once during the pebbling process, any valid AND forest will lead to an oracle design that achieves the minimum Toffoli count, as the number of Toffoli gates will match the number of nodes. In contrast, the Toffoli depth and ancillary qubit usage — key components of the space-time trade-off — are further influenced by the specific reversible pebbling strategy applied to the synthesized AND forest.

To better understand the space-time trade-off in the resulting oracle design, we explore how the structure of the AND forest influences these resource metrics. This analysis enables us to introduce additional constraints to the baseline SAT formulation, facilitating the construction of data-lookup oracles with distinct quantum resource configurations.

B. Additional Constraints for Exploring Space-time Trade-off

This subsection addresses two key questions: "What structural features of an AND forest minimize the number of pebbles required by a pebbling strategy?" and "What structural features minimize the number of steps needed to pebble it?" These questions are critical, as they directly correspond to data-lookup oracle constructions that minimize ancillary qubit usage and Toffoli depth, respectively.

To ensure the scalability of our SAT-based AND forest synthesis approach, we introduce two additional sets of constraints based on the principle of computing product terms incrementally, layer by layer. As outlined in Section III-A, this approach ensures that product terms with d variables ($2 \leq d \leq n$) are only computed once all $(d-1)$-variable terms have been determined and can serve as operands. Structuring the synthesis in this way reduces the process of generating an AND forest to solving $n - 2$ SAT instances, with each instance determining the fan-ins for the nodes in the d-th layer, which compute all d-variable product terms. This narrowing of possible fan-ins effectively reduces the solution space, enabling the construction of larger oracle designs while preserving accurate product term computation.

1) Minimizing Ancillary Qubit Count: As introduced in Section II-C, a pebble can only be placed on a node if both of its non-PI fan-in nodes are already pebbled. More than one pebble is required to pebble any valid AND forest. This is because no AND forest exists where all fan-ins of every node are always PIs — with two PIs as fan-ins, a node can only compute a product term involving two input variables.

We recognize that a more ancillary qubit-efficient pebbling strategy can be applied to an AND forest if every node has at least one fan-in that is a PI. When pebbling such an AND forest, the PI fan-in of each node can be omitted during the pebbling process. Then, each node can be pebbled as soon as its single non-PI fan-in is pebbled. This reduces the problem to finding a pebbling strategy for a collection of line graphs, where the longest line graph has a length of $n-1$, the one from

a PI to the node that computes the product term involving all n input variables. Existing research has shown that the length of the longest line graph determines the minimum number of pebbles required to pebble it, specifically, $\lceil \log_2(n-2) \rceil + 1$ for $n \geq 4$ [17], which is, therefore, also the minimum number of pebbles required for such an AND forest.

Denote the set of indices of nodes in layer d as $\mathcal{L}_d$, for $i \in \mathcal{L}_d$, clauses

$$\bigvee_{k=1}^{n} c_{i,k} \text{ and } \bigvee_{k \in \mathcal{L}_{d-1}} c_{i,k} \tag{7}$$

ensure each node in layer d has one fan-in that is a PI and the other that is a node in layer $d-1$.

Such AND forests always exist. As discussed in Section III-A, any product term involving i input variables ($2 \leq i \leq n$) can be computed by applying a Boolean AND operation to a product term involving $i-1$ of those variables and the remaining variable. Such a computing strategy guarantees that every node in the DAG has at least one fan-in that is a PI, ensuring the desired structure.

To ensure that the synthesized AND forest exhibits this structural feature, we introduce the following clause as an additional constraint to the baseline SAT formulation. For each node n_i, the constraint is defined as:

$$\bigvee_{j=1}^{n} c_{i,j}. \tag{8}$$

This constraint guarantees that every node in the synthesized AND forest has at least one PI as a fan-in, ensuring the applicability of the pebbling strategy described above, leading to ancillary-qubit-efficient data-lookup oracle designs.

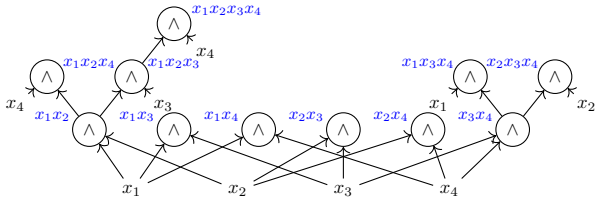
2) Minimizing Toffoli Depth: It is empirically observed that the reversible pebbling strategies for an AND forest tend to involve fewer steps when the AND trees comprising the forest have shallower depths. Based on this observation, we propose the following intuitive argument: when minimizing pebbling steps, an AND forest is preferable if it has more POs, i.e., nodes without fan-out. Thus, an AND forest with more POs consists of more shallow AND trees, resulting in a pebbling strategy that requires fewer steps.

To implement this constraint, we introduce an additional type of helper variable into the SAT formulation. For each node n_i, we define a *non-PO variable* o_i to indicate whether the node has a fan-out ($o_i = 1$) or not ($o_i = 0$). This value is determined by:

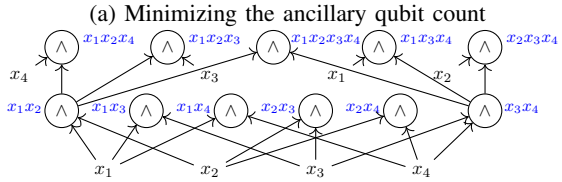
$$o_k = \bigvee_{i=k+1}^{2^n} c_{i,k}. \tag{9}$$

When seeking an AND forest with d POs, where d is an integer constant, meaning there are $2^n - n - 1 - d$ non-PO nodes, the following Boolean cardinality constraint is introduced:

$$\sum_{k=n+1}^{2^n} o_k = 2^n - n - 1 - d. \tag{10}$$



(a) Minimizing the ancillary qubit count



(b) Minimizing the Toffoli depth

Fig. 3: Synthesizing AND forests of different structures for constructing 4-variable data-lookup oracles. The product term computed at each node is marked in blue.

In our implementation, we use the *sequential counter encoding* proposed in [18] to handle this constraint. By incrementally targeting a larger d and solving the corresponding SAT instances, the first satisfiable instance corresponds to the desired AND forest.

In Fig. 3, we illustrate the AND forests synthesized for the construction of 4-variable data-lookup oracles by solving the corresponding SAT instances, encoded according to our SAT formulation. Specifically, the SAT instance for the AND forest in Fig. 3a incorporates the additional constraints in Section IV-B1, aiming to find the AND forest corresponding to the ancillary qubit-count-minimum design. The SAT instance for the AND forest in Fig. 3b applies the second genre of additional constraints, seeking to find the AND forest yielding the Toffoli-depth-minimum oracle design without constraints on the ancillary qubits. The resource requirements of the resulting oracle designs are discussed in Section VI. By applying the modified reversible pebbling game to these two AND forests, we observe that while the first design results in a data-lookup oracle that requires 12 Toffoli gates and is with a Toffoli depth of 11, it uses only two ancillary qubits, whereas the second design uses 11 Toffoli gates and achieves a Toffoli depth of 4 but requires five ancillary qubits. These results are consistent with expectations and demonstrate the trade-offs between qubit count and Toffoli depth.

It is evident that constructing an ancillary qubit-efficient data-lookup oracle typically comes at the cost of a higher Toffoli count and depth, as many nodes in the AND forest must be uncomputed and recomputed during the pebbling process due to the limited availability of ancillary qubits. On the other hand, a low-Toffoli-depth oracle construction often corresponds to a relatively small Toffoli count. However, this relationship is not deterministic, as the Toffoli depth also depends on whether multiple Toffoli gates are operand-independent and can be executed in parallel.

V. Heuristic Constraints for AND Forest Synthesis

This section introduces additional heuristic constraints for the SAT formulation presented in Section IV-A. These constraints serve as alternatives to those proposed in Section IV-B, providing more targeted guidance on the structural features of the AND forest. While these heuristic constraints do not precisely match the optimal solution space defined by the exact constraints, they significantly reduce the solution space, enhancing the scalability of our SAT-based data-lookup oracle construction method.

The core idea, regardless of whether the focus is on minimizing the ancillary qubit count or the Toffoli depth, is to compute product terms level by level. As introduced in Section III-A, this means that product terms containing i variables ($2 \leq i \leq n$) are only computed once all terms with $i-1$ variables have been calculated and can serve as operands.

A. Minimizing Ancillary Qubit Count

As discussed in Section IV-B1, minimizing ancillary qubit count requires computing each product term containing i variables by applying a Boolean AND operation between a term with $i-1$ variables and the remaining variable. The design space lies in selecting which of the $\binom{n}{i-1}$ terms should contribute to the computation of all i-variable terms, to maximize the number of nodes without fan-out (i.e., the POs of the AND forest). Maximizing the number of PO nodes is important because, as explained in Section IV-B2, AND forests with more PO nodes allow for reversible pebbling strategies that involve fewer steps, thus reducing Toffoli depth.

By following this approach, the oracle construction is decoupled into multiple sub-problems with a significantly constrained solution space, as the selection of product terms for each layer is independent of other layers. Each layer's decision-making problem can be formulated as a Boolean cardinality problem and efficiently solved using a SAT solver.

B. Minimizing Toffoli Depth

Our strategy for minimizing the Toffoli depth of the resulting AND forest also focuses on the computation of product terms layer by layer. For product terms involving an even number d of variables, the operands are selected from terms with $\frac{d}{2}$ variables, i.e., nodes in layer $\frac{d}{2}$. For odd values of d, the operands are selected from $(d-1)$-variable terms, similar to the approach used for minimizing ancillary qubit count. The rationale behind this approach is that once the depth of nodes for even-d-variable terms is minimized ($\lceil \log_2 d \rceil$), the depth for the subsequent layer is automatically minimized to $\lceil \log_2 d \rceil + 1$. This ensures that the depth of the synthesized AND forest is minimized, thereby reducing the Toffoli depth. This constraint can be realized similarly to Eq. (7).

The problem of maximizing the number of POs in the AND forest can now be framed as a stepwise decision-making process, with one step for each layer d. A slight distinction arises depending on whether d is odd or even. When d is odd, the objective is to maximize the number of POs among the nodes in layer $d-1$. This can be achieved by setting the number of POs among these nodes to a specific integer e using the following constraint in the SAT formulation:

$$\sum_{k \in \mathcal{L}_{d-1}} o_k = \binom{n}{d-1} - e. \tag{11}$$

By incrementally increasing e and solving the corresponding SAT instances, the first satisfiable instance provides the desired fan-in configuration for the nodes in layer d, maximizing the number of POs in layer $d-1$. When d is even, the goal remains to maximize the number of POs among nodes in layer $\frac{d}{2}$. However, some nodes in layer $\frac{d}{2}$ may already be designated as non-POs, as they contribute to the computation of $(\frac{d}{2}+1)$-variable terms. Denoting the number of such nodes as e', this adjustment requires subtracting e' from e in Eq. (11). Despite this adjustment, the decision-making problem remains a Boolean cardinality problem.

VI. Experimental evaluation

In this section, we evaluate the proposed data-lookup quantum oracle construction method. As outlined in Section III-B, for a given target function with n variables, we construct the data-lookup oracle in two steps: First, we synthesize an AND forest (Section III-B1) that guides the computation of each product term. This is achieved by solving a SAT instance based on the formulation presented in Section IV-A, augmented with the additional constraints (Section IV-B, denoted as '(E)') or heuristic constraints (Section V, denoted as '(H)'). Second, we compile the computation of these product terms into a quantum circuit by applying the modified reversible pebbling game (Section III-B2) to the synthesized AND forest. Our implementation builds upon the open-source quantum oracle synthesis library `caterpillar` and has been released as open source[1].

The experiments were conducted on a Qualcomm Snapdragon X Plus processor with 16 GB of memory. `MiniSat` [19] is adopted as the SAT solver. The timeout for each two-stage oracle construction task is set to two hours.

We evaluate the quantum resources required by different data-lookup oracle constructions. Table I presents each oracle's construction method ('Construction'), variable size ('#Vars'), Toffoli count ('#Toffoli'), Toffoli depth ('Depth'), ancillary qubit count ('#Qubits'), the number of CNF clauses accumulated during SAT-based AND forest synthesis ('#Clauses'), and construction time in seconds ('Time'). Construction times less than 0.005s are rounded down to 0.00s in the table.

When using exact constraints, our SAT-based AND forest synthesis approach struggles to scale and fails to synthesize a design within the runtime budget for more than four variables. However, with heuristic constraints, the approach successfully constructs oracles for up to seven variables — practical for real-world applications, such as [8]. Although the sample size is limited, it is noteworthy that for the 4-variable case, the resource efficiency of the designs generated using

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

TABLE I: Quantum resources required by different constructions of data-lookup oracles with various variable size.

Construction	#Vars	#Toffoli	Depth	#Qubits	#Clauses	Time [s]
Ancillary qubit (E)	4	12	11	2	9 320	0.02
Toffoli depth (E)	4	11	4	5	4 546	0.00
Ancillary qubit (H)	4	12	11	2	1 763	0.00
Toffoli depth (H)	4	11	4	5	1 785	0.00
Sawtooth circuit	4	14	14	2	N/A	N/A[†]
Ancillary qubit (E)	5	N/A	N/A	N/A	N/A	T/O[‡]
Toffoli depth (E)	5	N/A	N/A	N/A	N/A	T/O
Ancillary qubit (H)	5	31	20	3	15 598	0.03
Toffoli depth (H)	5	27	8	9	29 053	0.01
Sawtooth circuit	5	30	30	3	N/A	N/A
Ancillary qubit (H)	6	70	51	3	119 884	13.34
Toffoli depth (H)	6	58	11	26	282 036	0.63
Sawtooth circuit	6	62	62	4	N/A	N/A
Ancillary qubit (H)	7	192	113	4	806 346	249.33
Toffoli depth (H)	7	125	8	42	2 520 720	6433.71
Sawtooth circuit	7	126	126	5	N/A	N/A

[†] The number of CNF clauses and runtime does not apply to sawtooth circuit-based constructions, as they are generated deterministically.
[‡] Time out.

heuristic constraints matches that of designs generated using exact constraints. This demonstrates that, while the heuristic constraints significantly prune the solution space to improve scalability, they still approximate the optimal solution space closely, offering practical scalability.

The ancillary-qubit-count-oriented constructions ('ancillary qubit') achieve the most efficient ancillary qubit usage across all variable sizes. As discussed in Section IV-B1, our constructions exhibit logarithmic growth in ancillary qubit count with increasing variable size, compared to the linear growth seen in the sawtooth-circuit-based designs. Hence, by exploiting the introduced flexibility in the space-time trade-off, our approach can effectively facilitate constructing oracles that are better suited to quantum systems with limited qubit resources.

Our Toffoli-depth-oriented constructions ('Toffoli depth') also demonstrate superior performance. Optimizing for Toffoli depth generally also leads to a reduction in Toffoli count. In all evaluated cases, the Toffoli-depth-oriented constructions are also the most Toffoli-efficient designs. This is because our approach leverages ANF product terms, whereas the SoTA design relies on enumerating minterms. Specifically: (1) Product terms do not involve all variables, many to be computed in parallel without data contention, whereas minterms include all variables, requiring sequential computation in sawtooth-circuit-based constructions. (2) Computing product terms requires fewer Boolean AND operations — and thus fewer Toffoli gates — than computing minterms.

One might wonder why our constructions do not always achieve the theoretical minimum Toffoli count proposed in Section III-A, i.e., $2^n - n - 1$. For instance, in the 5-variable and 7-variable cases, the number of Toffoli gates slightly exceeds the minimum (27 vs. 26 and 125 vs. 120, respectively). This discrepancy arises because the minimum Toffoli count can only be guaranteed by adding a constraint to the modified reversible pebbling game, ensuring that each node in the AND forest is pebbled exactly once. However, in our approach, we allow each node to be computed at least once, which enables a fuller exploration of the space-time trade-off. This flexibility in node computation explains the minor differences in Toffoli count.

Although the construction time for the proposed approach may appear excessive, particularly for the 7-variable Toffoli-depth-oriented design, it is important to note that the majority of a data-lookup oracle corresponds to computing all possible product terms over the input variables. This structure is determined solely by the variable size and is reused across all Boolean functions of the same size. The specific function only determines which subset of these computed terms contributes to the output. Thus, the oracle construction can be performed once and reused for any function with the same input size, amortizing the synthesis cost.

VII. Conclusion

In this paper, we proposed a novel approach to constructing data-lookup quantum oracles by leveraging the *algebraic normal form* (ANF) of Boolean functions. This approach offers two key contributions: (1) Compared to traditional truth-table-based constructions, it achieves an $\mathcal{O}(n)$ reduction in the minimum number of Toffoli gates required for an n-variable data-lookup oracle; (2) It formulates the data-lookup oracle construction as a two-stage design automation problem, introducing and exploiting a space-time trade-off to create oracles with varying quantum resource requirements, providing flexible solutions based on the resource constraints of the target quantum system. This flexibility in design is not supported by the state-of-the-art sawtooth-circuit-based construction. These advancements enable more resource-efficient quantum circuit implementations, particularly in the context of *fault-tolerant quantum computing* (FTQC) systems, where Toffoli count and depth are critical metrics. Future work could explore extending this approach to larger-scale oracle designs.

Acknowledgment

The authors would like to thank the anonymous reviewers for their constructive feedback and suggestions. This work was partially supported by Synopsys Inc.

References

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997.

[2] N. Wiebe, A. Kapoor, and K. M. Svore, "Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning," *Quantum Information & Computation*, vol. 15, no. 3–4, pp. 316–356, 2015.

[3] G. Meuli, M. Soeken, and G. De Micheli, "Xor-and-inverter graphs for quantum compilation," *npj Quantum Information*, vol. 8, no. 7, 2022.

[4] G. Meuli, M. Soeken, E. Campbell *et al.*, "The role of multiplicative complexity in compiling low T-count oracle circuits," in *IEEE/ACM International Conference on Computer-Aided Design*, 2019.

[5] R. Babbush, C. Gidney, D. W. Berry *et al.*, "Encoding electronic spectra in quantum circuits with linear T complexity," *Physical Review X*, vol. 8, no. 4, 2018.

[6] G. H. Low, V. Kliuchnikov, and L. Schaeffer, "Trading T gates for dirty qubits in state preparation and unitary synthesis," *Quantum*, vol. 8, p. 1375, 2024.

[7] T. Häner, V. Kliuchnikov, M. Roetteler *et al.*, "Space-time optimized table lookup," 2022.

[8] R. Krishnakumar, M. Soeken, M. Roetteler *et al.*, "A Q# implementation of a quantum lookup table for quantum arithmetic functions," in *IEEE/ACM International Workshop on Quantum Computing Software (QCS)*, 2022.

[9] D. Gottesman, "The Heisenberg representation of quantum computers," in *International Colloquium on Group Theoretical Methods in Physics*, 1998, pp. 32–43.

[10] C. Gidney, "Halving the cost of quantum addition," *Quantum*, vol. 2, p. 74, 2018.

[11] K. Phalak, M. Alam, A. Ash-Saki *et al.*, "Optimization of quantum read-only memory circuits," in *IEEE International Conference on Computer Design*, 2022, pp. 117–123.

[12] S. Zhu, A. Sundaram, and G. H. Low, "Unified architecture for a quantum lookup table," 2024.

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

[14] G. Meuli, M. Soeken, M. Roetteler *et al.*, "Reversible pebbling game for quantum memory management," in *Design, Automation & Test in Europe Conference & Exhibition*, 2019, pp. 288–291.

[15] J. Boyar, R. Peralta, and D. Pochuev, "On the multiplicative complexity of Boolean functions over the basis ($\wedge$,$\oplus$,1)," *Theoretical Computer Science*, vol. 235, no. 1, pp. 43–57, 2000.

[16] A. M. Frisch and P. A. Giannaros, "SAT encodings of the at-most-k constraint: Some old, some new, some fast, some slow," in *International Workshop of Constraint Modelling and Reformulation*, 2010.

[17] E. Knill, "An analysis of Bennett's pebble game," *arXiv preprint math/9508218*, 1995. [Online]. Available: https://arxiv.org/abs/math/9508218

[18] C. Sinz, "Towards an optimal CNF encoding of Boolean cardinality constraints," in *Principles and Practice of Constraint Programming*, 2005, pp. 827–831.

[19] N. Sörensson and N. Een, "MiniSat v1.13 - a SAT solver with conflict-clause minimization," in *International Conference on Theory and Applications of Satisfiability Testing*, 2005.