

RESEARCH ARTICLE

Implementation of Grover's Iterator for Quantum Searching With an Arbitrary Number of Qubits

SIHYUNG LEE¹ AND SEUNG YEOP NAM², (Senior Member, IEEE)¹School of Computer Science and Engineering, Kyungpook National University, Daegu 41566, South Korea²Department of Information and Communication Engineering, Yeungnam University, Gyeongsan 38541, South Korea

Corresponding author: Sihyung Lee (sihyunglee@knu.ac.kr)

ABSTRACT Grover's algorithm harnesses the power of quantum computing to swiftly locate specific elements in an unstructured database, outperforming classical computers in tasks like database searching. This algorithm capitalizes on the unique ability of qubits to be in both 0 and 1 states simultaneously (superposition), allowing it to scan the entire search space at once. It then boosts the probability of the target element, making it more prominent. While the foundational concepts of Grover's algorithm are well-documented, practical implementation using quantum operators, especially for large search spaces, remains less explored beyond basic examples with a small number of qubits. Existing general synthesis techniques often involve numerous operators or are time-consuming. Our proposed methods specifically address the amplitude-amplification component of Grover's algorithm for any size of search space. These methods detail the required types and quantities of qubits and operators, emphasizing minimal usage and efficient assembly. Developed and evaluated in Python, our methods consistently identified target elements with over 95% accuracy and achieved configurations comparably compact as those in the existing literature but at a faster pace. We anticipate that these methods facilitate practical implementations of Grover's algorithm across various domains.

INDEX TERMS Grover's algorithm, program synthesis, quantum algorithm, quantum computing, quantum program.

I. INTRODUCTION

Quantum computers operate on a fundamentally different basis than classical computers, with the potential to tackle challenges that are currently tough for classical computers. Their key distinction lies in using qubits for data processing. Unlike classical bits that are either 0 or 1, qubits can exist in a state of superposition, representing both 0 and 1 along with their respective probabilities [1]. This unique feature allows quantum computers to process multiple possibilities at once, offering faster solutions for various problems than classical computers. However, today's quantum machines are still in their early stages, handling a few hundred qubits and facing higher error rates compared to classical computers [2]. Despite these limitations, advancements in quantum technology are ongoing, and it is anticipated that quantum

computers will eventually match or surpass the performance and reliability of classical computers in the future [3].

As quantum computing technology progresses, researchers have developed several algorithms that exploit the unique properties of quantum computers. Grover's algorithm [4] is one notable example. It addresses the challenge of search through unstructured data, offering a significant speed advantage over classical computer solutions. The algorithm achieves this by creating an equal superposition of states across a group of qubits, allowing it to examine the entire search space simultaneously. It then selectively enhances the probability of the desired outcome, making it more easily identifiable among other possibilities. With the advent of faster and more dependable quantum hardware, Grover's algorithm is poised to accelerate a range of applications, including database searches, cryptanalysis, pattern recognition, quantum simulations, and more [5].

While existing research explains the mechanics of Grover's algorithm and its basic steps through mathematical equations,

The associate editor coordinating the review of this manuscript and approving it for publication was Hamdi Abdi.

these studies primarily demonstrate its implementation using quantum operators for small search spaces, limited to a few qubits [5], [6], and do not detail how to scale it to larger spaces. Extending these implementations to a larger search space is not straightforward from these small-scale examples. This complexity arises partly because the operators used on quantum computers, such as Hadamard and phase shift [3], manipulate qubits in line with quantum physics principles, which are often perceived as counterintuitive. Therefore, it is challenging to understand their roles and to predict the outcomes when multiple operators are employed in sequence [7]. Furthermore, adding one qubit doubles the number of states that a quantum program needs to consider and manipulate. The number of such potential quantum programs also grows exponentially with the addition of qubits. Consequently, as more qubits are required, it becomes increasingly demanding to both find the right combination of operators that implement a desired function and to optimize the implementation [8], [9].

Alternatively, general methods for implementing any quantum algorithm can be applied [8]. Some of these methods progressively align operators to the required configuration using set rules [10], [11], [12], [13], [14], [15], [16]. These methods can quickly produce implementations, but they often lead to an exponential increase in the number of operators as the qubit count grows [17]. Other methods pursue a broader exploration of potential implementations, which can result in configurations with fewer operators [18,7,19,20,21,22]. However, this exploration is time-consuming, making it feasible only for systems with a relatively small number of qubits in a practical timeframe. For example, recent synthesis frameworks, such as QSyn [21] and QFAST [22], can synthesize quantum programs with up to 5 qubits but fail to synthesize a 6-qubit program within an hour. The primary reason for this limitation is the exponential growth of the search space and, correspondingly, the exploration time as the number of qubits increases [23]. This remains the case even after applying their optimization techniques.

We introduce methods for implementing a crucial part of Grover's algorithm, applicable to any size of search space. This key part, amplitude amplification, selectively boosts the likelihood of achieving the desired result and is suitable for various search objectives. We demonstrated that the implementations generated by our methods successfully identify the intended outcomes in over 95% of cases. Additionally, the proposed methods efficiently generate implementations where the number of operators increases linearly with the qubit count representing the search space.

In Table 1, we summarize the differences between previous studies and our proposed methods. We also list our main contributions as follows.

- We offer guidance on applying Grover's algorithm to an arbitrary size of search space, detailing the numbers and types of qubits and operators needed, and instructions for assembling these elements. This includes strategies for optimizing the process, such as reducing the number

TABLE 1. Differences between previous studies and proposed methods.

Classification	Synthesis Time	# of Operators	Scalability
Manual implementations [5,6,27] are provided for 2–3 qubits without detailing how to scale them to larger qubit counts.	N/A	Low	2–3 qubits
Rule-based techniques [10–16,34,35] progressively adds operators to the required configuration using set rules. These are quick, but the number of operators grows exponentially.	Low–Mid	High	2–20 qubits
Search-based techniques [7,18–22] explore a more diverse space of potential implementations to find a compact one, but the exploration is time consuming.	High	Low	2–4 qubits
Proposed Methods are specifically tailored to optimize the implementation of Grover's algorithm, and thus are quick and use few operators.	Low	Low	2–100 qubits

of operators and stages through exploiting the global phase difference and efficiently reusing ancilla qubits.

- We tested our methods across various search space sizes, consistently achieving over 95% accuracy in identifying desired outcomes. Compared to existing methods, our approach requires moderate resources (with the number of operators growing linearly with the qubit count), slightly more than the minimum used by existing methods, yet our methods produce implementations more quickly.
- We developed a synthesizer in Python, making the code and Grover's algorithm implementations available online for other researchers to review and utilize.¹ We have made available our synthesizer codes along with sample results.

The paper is organized as follows. Section II covers quantum programming fundamentals and Grover's algorithm, along with existing methods related to ours. Section III details our proposed methods, including the representation of target functions and the implementation with quantum operators. Section IV assesses our methods against current practices. Section V concludes the study and outlines future research directions. Additionally, we include a glossary of terms used in the paper, as shown in Table 2.

II. BACKGROUND AND RELATED WORK

In this section, we cover the fundamentals of quantum programming (Section II-A), Grover's algorithm (Section II-B), and related work (Section II-C). Readers already acquainted with quantum programming and Grover's algorithm may prefer to begin directly with Section II-C.

¹On our GitHub repository at <https://github.com/sihyunglee26/Grover-Amplitude-Amplification-23/tree/main>

TABLE 2. Summary of terms.

Term	Description
<i>CCNOT</i>	(Controlled-Controlled-NOT Operator) Also known as the Toffoli gate, this three-qubit operator flips the state of the third qubit from 0 to 1 (and vice versa) only when the first two qubits are in the 1 state.
<i>CZ</i>	(Controlled-Z Operator) A two-qubit operator that inverts the phase of the second qubit only when both qubits are in the 1 state
<i>H</i>	(Hadamard Operator) A single-qubit operator that generates a balanced superposition, equally weighting the 0 and 1 states
<i>I</i>	(Identity operator) This represents the absence of any operation, leaving the qubit's state unchanged.
<i>N</i>	(Search Space Size) The total number of items in the search space
<i>n</i>	(Qubit Count) The number of qubits needed to represent <i>N</i> items, calculated as $n = \lceil \log_2(N) \rceil$
<i>O*</i>	(Conjugate Transpose Operator of O) This symbolizes the conjugate transpose, or adjoint, of a given quantum operator <i>O</i> .
<i>R_N</i>	(Reflection Operator in Grover's Algorithm) A component in Grover's algorithm that inverts the states of <i>N</i> items around their average value, thus selectively enhancing the amplitude of the targeted search item

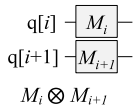
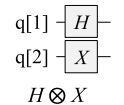
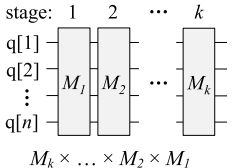
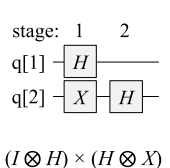
A. BACKGROUND ON QUANTUM PROGRAMS

In a classical computer, a bit serves as a fundamental unit of data. A quantum computer employs a similar concept known as a qubit. However, unlike classical bits, which can only be 0 or 1, a qubit can exist in a superposition of both 0 and 1 simultaneously [1]. This means that it can have a certain probability of being 0 and another probability of being 1, while also having specific phases associated with each state. Upon measurement of the qubit, its value collapses to either 0 or 1, determined by the assigned probabilities.

To represent such a qubit's state, we use a vector composed of two complex numbers s_0 and s_1 , as shown in Notation 1 in Table 3. The squares of the absolute values of these complex numbers, $|s_0|^2$ and $|s_1|^2$, respectively, represent the probabilities of the qubit being 0 and 1, such that $|s_0|^2 + |s_1|^2 = 1$. Additionally, the angles they form with the positive real axis correspond to the phases associated with each state. For instance, $(1/\sqrt{2}, 1/\sqrt{2})$ indicates equal probabilities 1/2 of the qubit being 0 and 1 with the same phase. On the other hand, $(1/\sqrt{2}, -1/\sqrt{2})$ represents equal probabilities, but with phases differing by π radians. These unique properties of qubits allow quantum computers to perform complex computations more efficiently in certain scenarios compared to classical computers, and the Grover's algorithm is one such example.

When multiple qubits are present, their joint state is expressed using the Kronecker product with symbol \otimes . In Table 3, Notation 2 illustrates the combined state of two qubits: (s_0, s_1) and (t_0, t_1) . The Kronecker product results in four terms: s_0t_0, s_0t_1, s_1t_0 and s_1t_1 , which respectively represent the probabilities and phases of the qubits being '00',

TABLE 3. Notations for quantum programs.

ID	Notation	Example
1	$\begin{bmatrix} s_0 \\ s_1 \end{bmatrix}$	$\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$ $\begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$
2	$\begin{bmatrix} s_0 \\ s_1 \end{bmatrix} \otimes \begin{bmatrix} t_0 \\ t_1 \end{bmatrix} = \begin{bmatrix} s_0t_0 \\ s_0t_1 \\ s_1t_0 \\ s_1t_1 \end{bmatrix}$	$\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \otimes \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} 1/2 \\ -1/2 \\ 1/2 \\ -1/2 \end{bmatrix}$ $H: \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$
3	$\begin{bmatrix} m_{00} & m_{01} \\ m_{10} & m_{11} \end{bmatrix} \times \begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} s'_0 \\ s'_1 \end{bmatrix}$	$X: \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$ $Z: \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \times \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$
4		
5		

'01', '10' and '11'. For example, let us consider two qubits represented as $(1/\sqrt{2}, 1/\sqrt{2})$ and $(1/\sqrt{2}, -1/\sqrt{2})$. Their joint state using the Kronecker product becomes $(1/2, -1/2, 1/2, -1/2)$ indicating equal probabilities of 1/4 for the qubits to be '00', '01', '10', and '11', respectively. The phases associated with these values are 0, π , 0, and π , respectively. In a similar manner, when dealing with *n* qubits, their collective state is represented by a vector comprising 2^n complex numbers. This vector captures the probabilities and phases associated with the various quantum states of the combined qubit system.

We now delve into the operations on qubits. A quantum operation on a single qubit manipulates the qubit's state by adjusting its probabilities and phases. This operation is represented mathematically by the multiplication of a 2×2 unitary matrix [24], as illustrated in Notation 3 in Table 3. Accordingly, the operation ensures that the probabilities of the qubit's states add up to 1 after manipulation. Also, an inverse operation can be applied through the conjugate transpose of the original matrix. Table 3 showcases three fundamental quantum operations: Hadamard (*H*), NOT (*X*) and Pauli-Z (*Z*). *H* evenly distributes the probabilities into states 0 and 1, creating a superposition; *X* interchanges the probabilities and phases of states 0 and 1; and *Z* reverses the sign of state 1, while maintaining the state 0 as it is.

Operations on multiple qubits are expressed using the Kronecker product of the corresponding matrices, following a similar approach as how multiple qubits are represented.

As such, operations on n qubits are represented by the multiplication of a $2^n \times 2^n$ unitary matrix. In Table 3, Notation 4 illustrates operations M_i and M_{i+1} performed on two qubits i and $i+1$, respectively. Their joint effect is expressed as $M_i \otimes M_{i+1}$. The first example on the right demonstrates the application of H on qubit 1 and X on qubit 2, resulting in the combined operation $H \otimes X$. In the second example, no operation is applied to qubit 1, and H is applied to qubit 2, resulting in the combined operation $I \otimes H$. I represents the identity matrix, which is equivalent to no operation being applied.

Finally, quantum programs (or algorithms) consist of a sequence of operations applied to qubits, one after another. When applying a single operation, its corresponding matrix is multiplied with the qubits' state. As a result, when multiple operations are applied in series, their corresponding matrices are multiplied in reverse order. In Table 3, Notation 5 illustrates the application of k consecutive operations, M_1 through M_k , on n qubits. M_1 is applied first, while M_k is the last operation in the series. The operations performed simultaneously on the qubits form individual stages, with each operation M_i representing one stage. The total number of stages in the program is thus k . The joint effect of applying these k stages is the product of the corresponding matrices in reverse order, represented as $M_k \times \dots \times M_2 \times M_1$. For example, the program shown on the right demonstrates the application of two stages: $H \otimes X$ followed by $I \otimes H$, on 2 qubits. The program, as a whole, performs the operation $(I \otimes H) \times (H \otimes X)$. In most cases, the final stage of the program involves measurement, which observes the qubits and provides computational results. While not explicitly shown in the example, measurement is a typical step in quantum programs to extract meaningful information from the quantum state.

B. GROVER'S ALGORITHM

Grover's algorithm [4] harnesses quantum computers to solve general search problems more efficiently than classical computers. The algorithm is designed to locate a specific item that meets a given condition among N items. It operates under the assumption that the N items are randomly distributed, with no known relationships regarding their relative positions. In classical computing, on average, a search for the target item would require exploring $N/2$ items, leading to a time complexity proportional to N . However, Grover's algorithm takes advantage of quantum computing and reduces the search time to be proportional to \sqrt{N} . This speedup is achieved by exploiting quantum superposition.

Fig. 1 presents an overview of Grover's algorithm. The algorithm represents N items with $n = \lceil \log_2(N) \rceil$ qubits, such that each combination of the qubits' values corresponds to one distinct item. For examples, $N = 4$ items are encoded into $n = 2$ qubits, where the qubits being '00' through '11' represent items 1 through 4, respectively. In Fig. 1, the n qubits are termed as data qubits. Additionally, the algorithm employs ancilla qubits, which, while not representing data, aid in the algorithm's execution. In step 1, the algorithm allocates

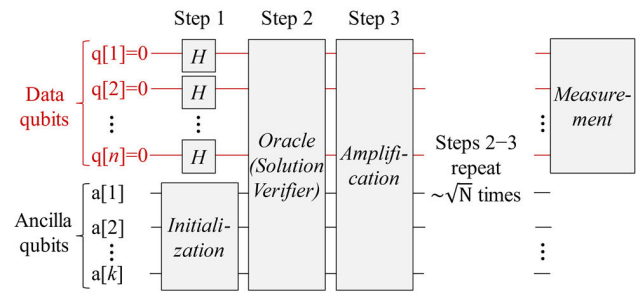


FIGURE 1. Overview of Grover's algorithm.

identical probabilities to each of the N items by forming a superposition with H . The subsequent steps, 2 and 3, increase the likelihood of the target item and diminish those of the rest. By consistently applying these steps proportional to \sqrt{N} times (i.e., $\lfloor \pi \sqrt{N}/4 \rfloor$ times [25]), the probability of the target item surpasses 90%. Finally, upon measuring the data qubits, there is a more than 90% probability of selecting the target item. Repeating the procedures and taking measurements allows us to accurately identify the target item with a high degree of certainty.

We now detail the functions of the three steps. In step 1, the qubits undergo a one-time setup process. The data qubits are placed into a superposition with equal probabilities. Fig. 2(a) illustrates the condition of data qubits following the completion of step 1, with 4 items ($N = 4$) encoded using 2 qubits ($n = 2$). Each of the four items is in an equal superposition with an amplitude of $1/2$, giving them all an identical probability of $(1/2)^2 = 1/4$. The ancilla qubits also undergo initialization procedures, and the specific configurations depend on the implementations of steps 2 to 3.

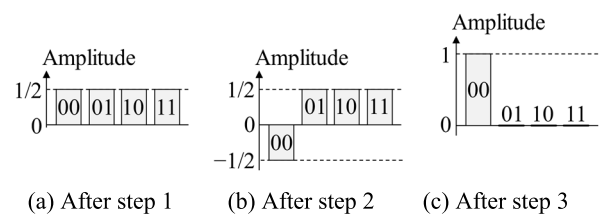


FIGURE 2. State of data qubits following stages 1 to 3 for $N = 4$ items encoded with $n = 2$ qubits, with '00' as the target state.

In step 2, the target item undergoes a sign inversion. For example, as in Fig. 2(b), if the target is represented by '00' and has a state of $1/2$, its state is transformed into $-1/2$, equivalent to a phase shift of π radians. All other states are unaffected. This inversion serves to highlight the target item among the others, enabling step 3 to subsequently amplify its probability. This second step is termed the "oracle" because it designates whether an item is the target. One might question: if we can already identify the target, why not directly retrieve it? Verifying a potential solution is often simpler than discovering it. Let us consider the prime factorization of a positive integer, m . Given prime numbers p and q , it is

straightforward to confirm that $p \times q = m$, yet more challenging to identify these numbers initially. The oracle, leveraging the superposed input states, examines all items at once and signifies the target by inverting its sign.

In step 3, the amplitude of the item with the inverted sign is enhanced, while the amplitudes of the others are reduced. This is accomplished by reflecting the states about their average value. Fig. 2(c) depicts the amplitudes after applying step 3 to the conditions shown in Fig. 2(b). The average amplitude across the four items is computed as $(-1/2 + 1/2 + 1/2 + 1/2) / 4 = 1/4$. Since the target item's state is sign-inverted to $-1/2$, the average is far from the target and slightly less than those of non-target items. Consequently, flipping the states about the average amplifies the target item's amplitude $(1/4 + (1/4 - (-1/2))) = 1$, and reduces the non-target items' amplitudes closer to zero $(1/4 - (1/2 - 1/4) = 0)$.

This paper concentrates on the execution of step 3 for a system comprising an arbitrary count of qubits, denoted as n . Implementing step 1 is relatively simple: each qubit is initially set to 0, followed by the application of an H to each. The configuration of step 2 is contingent upon the specific search problem being addressed, as different problems necessitate unique solution criteria and, consequently, distinct verification processes. Although step 3's methodology is consistent across a range of problems, devising a universal implementation for an unspecified number of qubits is not straightforward.

C. RELATED WORK

This paper focuses on the implementation of step 3 (amplitude amplification) in Grover's algorithm; consequently, we review the literature related to this step. There are also existing studies on the implementation of step 2 (the oracle) for various applications, such as cryptanalysis [26].

Previous literature offers implementations for step 3 that are tailored to specific numbers of qubits [5], [18], [27]. For instance, Fig. 3 illustrates two distinct approaches to carrying out step 3 for $n = 2$ qubits. However, these examples lack a method for extending the implementations to accommodate a variable number of qubits. The foundational paper on Grover's algorithm [4] suggests that step 3 can be broken down into a sequence of operations: $H^{\otimes n} \times D \times H^{\otimes n}$, with D representing a diagonal matrix. Yet, it does not detail how to operationalize D . Building on this notion, our research provides a method to implement D , regardless of the qubit count n . This proposed approach requires operators and stages that scale with n .

Other research offers techniques to implement a diagonal matrix by employing controlled operators designed for 2 and 3 qubits [28]. Our approach, while similarly employing controlled operators, is fine-tuned for the specific requirements of Grover's algorithm. We streamline the diagonal matrix, such that it is easier to implement, while retaining full functionality through repeated applications during step 3. We also

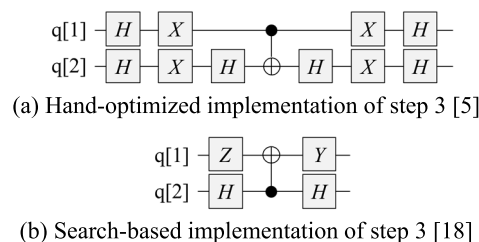


FIGURE 3. Sample implementations of stage 3 for $n = 2$ qubits.

identify controlled operators suitable for this matrix, specifically CZ and $CCNOT$ operators, and recommend strategies for reducing the number of ancillary qubits needed by reusing them wherever feasible. Furthermore, we demonstrate that our implementation method leads Grover's algorithm to correctly locate the target item with a high probability, as confirmed by simulations across a range of qubit numbers.

Various studies propose universal strategies for constructing any unitary matrix [10], [11], [12], [13], [14], [15], [16]. These approaches typically involve a consistent repertoire of operators arranged by predefined rules. Each operator placement incrementally closes the gap toward the desired matrix. For example, in the process depicted in Fig. 4, Shende et al. [13] break down an n -qubit unitary matrix into a combination of $(n-1)$ -qubit unitaries and controlled operations. This breakdown persists until the unitaries are simplified into 1 or 2 qubits in size, at which point they are translated into the available operator set [14]. In general, while these rule-based approaches can swiftly synthesize a unitary matrix, they tend to require a number of operators that grows exponentially with the number of qubits, n [17]. In contrast, our method is specifically tailored for Grover's algorithm, crafting its step 3 with operators that increases linearly with n .

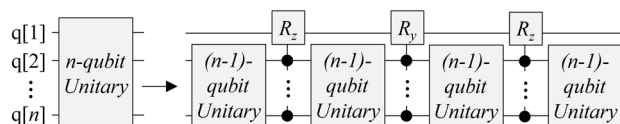


FIGURE 4. Example of rule-based synthesis of a unitary matrix.

To minimize operator counts in unitary-matrix synthesis, certain strategies scout through a multitude of potential implementations, favoring those that employ the fewest operators [18,7,19,20]. Given the vastness of the search area, these methods prioritize efficient ways to narrow the possible choices. For instance, some methods learn various potential implementations and leverage this knowledge to isolate a subset of operators that appears most efficient for constructing the desired matrix [18]. Alternatively, other methods adopt stochastic search techniques, such as ant colony optimization [7] and genetic algorithms [19], [20], which refine an initial program by randomly applying adjustments—adding, replacing, or reordering operators. The efficacy of each

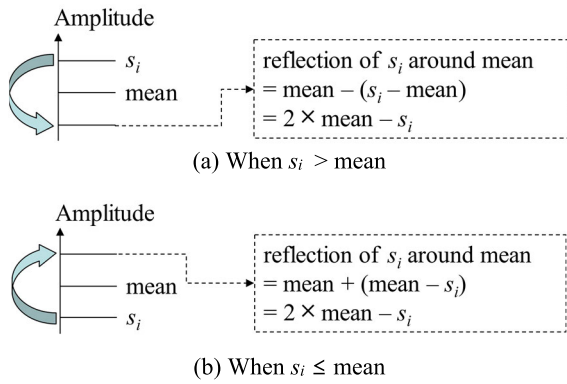


FIGURE 5. Reflection of state s around the mean.

adjustment, in terms of how closely the refined program conforms to the desired matrix, informs the likelihood of its selection in subsequent iterations. This process continues until the output aligns with the matrix.

Since search-based methods assess a wider array of implementations than rule-based counterparts, they can lead to more succinct solutions (e.g., the manual construction in Fig. 3(a) versus the search-derived result in Fig. 3(b)). However, the time required to traverse the possibilities increases, and the search-based techniques yield practical outcomes within a few hours for systems with fewer than 10 qubits [21], [22]. Furthermore, scaling the findings for systems with over 10 qubits is not straightforward. Our proposed method, in contrast, expedites the development of implementations for any large number of qubits, n , with the growth in operator count proportional to n . This not only speeds up the process but also ensures scalability for larger quantum systems.

III. SYNTHESIS METHODS FOR STEP 3

In this section, we delineate the process of executing step 3, known as the amplification step, of Grover's algorithm for any given number of qubits, n . We start by portraying step 3 as a unitary transformation (Section III-A) and proceed to demonstrate its realization using quantum operators (Section III-B).

A. UNITARY MATRIX REPRESENTATION OF STEP 3

Step 2 inverts the sign of the target item, setting the stage for its amplification in step 3. This amplification step involves a reflection operation where the quantum states of the items are mirrored about their collective mean value. This reflection operation affects the states differently, depending on their magnitude relative to the mean. We consider two cases: one where the state value, s_i , is greater than the mean, as illustrated in Fig. 5(a), and the other where s_i is less than or equal to the mean, as shown in Fig. 5(b). Regardless of the case, the reflection is described by the same transformation which results in a new state value of $2 \times \text{mean} - s_i$.

The mean of $N = 2^n$ states is $\sum_{k=1}^N s_k / N$, and thus the transformed state for s_i is expressed as $2 \times \text{mean}$

$- s_i = \{2s_1 + 2s_2 + \dots + 2s_{i-1} + (2 - N)s_i + 2s_{i+1} + \dots + 2s_{N-1} + 2s_N\} / N$. This transformation is achieved through multiplying the following unitary matrix R_N across N states ($s_1, s_2, \dots, s_{N-1}, s_N$). The diagonal elements of this matrix are $(2 - N) / N$, while the remaining elements are $2 / N$.

$$R_N = \begin{bmatrix} (2 - N)/N & 2/N & 2/N & \dots & 2/N \\ 2/N & (2 - N)/N & 2/N & \dots & 2/N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 2/N & 2/N & 2/N & \dots & (2 - N)/N \end{bmatrix}$$

For instance, if there are $N = 4$ items represented by $n = 2$ qubits, the corresponding reflection matrix and the altered states are as follows. The matrix's diagonal elements are $-2/4$, and the other elements are $2/4$. When this matrix multiplies each state s_i , the state is modified to be $2 \times \text{mean} - s_i$. As shown in Section II-B, this reflection amplifies the amplitude of the target item with inverted sign, simultaneously decreasing the amplitudes of the other items.

$$R_4 \times \begin{bmatrix} s_{00} \\ s_{01} \\ s_{10} \\ s_{11} \end{bmatrix} = \begin{bmatrix} -2/4 & 2/4 & 2/4 & 2/4 \\ 2/4 & -2/4 & 2/4 & 2/4 \\ 2/4 & 2/4 & -2/4 & 2/4 \\ 2/4 & 2/4 & 2/4 & -2/4 \end{bmatrix} \times \begin{bmatrix} s_{00} \\ s_{01} \\ s_{10} \\ s_{11} \end{bmatrix} \\ = \begin{bmatrix} 2 \times (s_{00} + s_{01} + s_{10} + s_{11}) / 4 - s_{00} \\ 2 \times (s_{00} + s_{01} + s_{10} + s_{11}) / 4 - s_{01} \\ 2 \times (s_{00} + s_{01} + s_{10} + s_{11}) / 4 - s_{10} \\ 2 \times (s_{00} + s_{01} + s_{10} + s_{11}) / 4 - s_{11} \end{bmatrix}$$

R_N qualifies as a unitary matrix because when R_N is multiplied by its conjugate transpose R_N^* , and vice versa, the result is the identity matrix I (i.e., $R_N \times R_N^* = R_N^* \times R_N = I$). In fact, R_N^* is equal to R_N , which means that applying R_N^* following R_N reverses each state back to its initial state. Consequently, the overall impact of this sequence results in the identity matrix I .

B. REALIZATION USING QUANTUM OPERATORS

The unitary transformation corresponding to step 3 of Grover's algorithm, denoted as R_N , can be factored into a product of three matrices. Mathematically, this is represented as $R_N = H^{\otimes n} \times D \times H^{\otimes n}$ [4]. Here, $H^{\otimes n}$ signifies the Kronecker product of n Hadamard matrices, expressed as $H \otimes H \otimes H \dots \otimes H$, and D stands for a diagonal matrix. Within matrix D , the leading diagonal element is 1 , while all other diagonal elements are set to -1 . The following equation illustrates the decomposition of the unitary matrix R_N as described.

$$R_N = H^{\otimes n} \times D \times H^{\otimes n} = H^{\otimes n} \times \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & -1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & -1 \end{bmatrix} \times H^{\otimes n}$$

In the decomposition of the matrix R_N , the term $H^{\otimes n}$ corresponds to the simultaneous application of n Hadamard operators to the n qubits, as depicted in Fig. 6. Therefore,

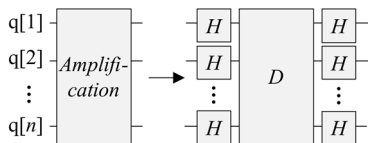


FIGURE 6. Realization of R_N as $H^{\wedge}(\otimes n) \times D \times H^{\wedge}(\otimes n)$.

the remaining task is to realize the implementation of the diagonal matrix D .

We opt for $-D$ instead of D because, despite being functionally similar, $-D$ can be implemented using fewer operators. The functional equivalence of D and $-D$ arises from the fact that substituting $-D$ for D still achieves reflection about the mean of the states, albeit with an additional inversion of their signs. As a result, the target state's amplitude remains amplified relative to the non-target states. Depending on the number of qubits, n , steps 2 and 3 can be applied again. Each time, the oracle in step 2 will reinvert the target state's sign, ensuring it consistently opposes the signs of the other states, which sets the stage for its amplification in step 3.

This type of functional similarity is known as equivalence under a global phase. When two matrices, M_1 and M_2 , differ by only a global phase factor (i.e., $M_1 = e^{i\theta} \times M_2$), they can be considered equivalent in quantum computing because their physical effects on a quantum state are often indistinguishable [9], [22]. However, this equivalence does not hold in every context and therefore needs to be verified before applying it.

In the matrix $-D$, the first element on the main diagonal is -1 , while all other diagonal elements are 1 . The effect of this operation is that the quantum state that represents all zeros across n data qubits (e.g., '00' for when $n = 2$) is sign-inverted, leaving all other states unaffected. For example, with $N = 4$ items encoded by $n = 2$ qubits, the matrix $-D$ and the resulting state modifications are as shown below. The state denoted by '00' undergoes a sign inversion, while the states '01', '10', and '11' remain as they are. Thus, implementing $-D$ requires a mechanism to single out the all-zero state and to exclusively invert its sign. We use the conditional operator $CCNOT$ to identify the all-zero state and CZ to invert its sign.

$$-D \times \begin{bmatrix} s_{00} \\ s_{01} \\ s_{10} \\ s_{11} \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} s_{00} \\ s_{01} \\ s_{10} \\ s_{11} \end{bmatrix} = \begin{bmatrix} -s_{00} \\ s_{01} \\ s_{10} \\ s_{11} \end{bmatrix}$$

In Table 4, we compile the methods for constructing the R_N matrix. The methods employ four quantum operators $\{H, X, CZ, CCNOT\}$, all of which are supported in many quantum computing platforms [6], [27]. The operators H and X act on single qubits, and they respectively generate a superposition and toggle between state 0 and state 1. The CZ operator works on a pair of qubits, and it applies the Z operation to the second qubit when the first qubit is in state 1. The Z operation inverts the sign of state 1, as described in Section II-A. The

TABLE 4. Algorithm for implementing step 3 of grover's algorithm.

```

01 # Amber: Keyword, Green: Variable
02 #
03 # N: Number of items to search for
04 # n: Number of qubits used to represent N items, [log2(N)]
05 # We assume that n > 1 (i.e., N > 2).
06 # q[1]-q[n]: Data qubits used to represent N items
07 # a[1]-a[n-1]: Ancilla qubits, which do not represent data,
08 # but aid in the algorithm's execution
09 # We assume that the ancilla qubits are initialized to 0.
10
11 # Synthesize step 3 with operators in {H, X, CZ, CCNOT}
12 function synthesize_reflection_around_mean(N, n, q[], a[])
13 # Step 1: Application of H^⊗n
14 for each i in [1, n], apply an H operator on q[i]
15
16 # Step 2: Application of -D
17 for each i in [1, n], apply an X operator on q[i]
18
19 apply a CCNOT operator on (q[1], q[2], a[1])
20 for each i in [3, n]
21     apply a CCNOT operator on (q[i], a[i-2], a[i-1])
22
23 apply a CZ operator on (a[n-1], q[n])
24
25 for each i in [3, n] in reverse order
26     apply a CCNOT operator on (q[i], a[i-2], a[i-1])
27 apply a CCNOT operator on (q[1], q[2], a[1])
28
29 for each i in [1, n], apply an X operator on q[i]
30
31 # Step 3: Application of H^⊗n
32 for each i in [1, n], apply an H operator on q[i]
    
```

$CCNOT$ operator, also known as the Toffoli gate, is a three-qubit operator that applies the X operation to the third qubit only when the first two qubits are both in state 1. Additionally, the methods require $n - 1$ ancilla qubits, which are initialized to the zero state at the start of the algorithm.

The methods for synthesizing R_N proceed in this manner: line 14 executes $H^{\otimes n}$, lines 17-29 perform $-D$, and line 32 applies $H^{\otimes n}$ again, culminating in the formation of $-R_N = H^{\otimes n} \times -D \times H^{\otimes n}$. Applying $H^{\otimes n}$ involves subjecting each of the n data qubits to an H operator (lines 14 and 32).

To enact $-D$, which targets the all-zero state for a sign inversion, we first transform this all-zero state to an all-one state by employing n X operators on the data qubits (line 17). This step facilitates the upcoming $CCNOT$ operators, which are designed to detect ones. Then, a series of $n - 1$ $CCNOT$ operators, aided by $n - 1$ ancilla qubits, are employed (lines 19-21) to flag the last ancilla qubit, $a[n - 1]$, with state 1 if all data qubits are in state 1. This triggers the CZ operator to flip the sign of the data qubits (line 23). The ancilla qubits are then reset to zero by reversing the $n - 1$ $CCNOT$ operations (lines 25-27), preparing them for reuse in future iterations of the process. The final step involves reapplying n X operators to the data qubits (line 29), reverting the all-one state to all-zero, thus restoring the correct configuration for the next sequence of steps 2-3.

Fig. 7 demonstrates the outcome of the procedure in Table 4, when applied to a system with $n = 4$ qubits. The

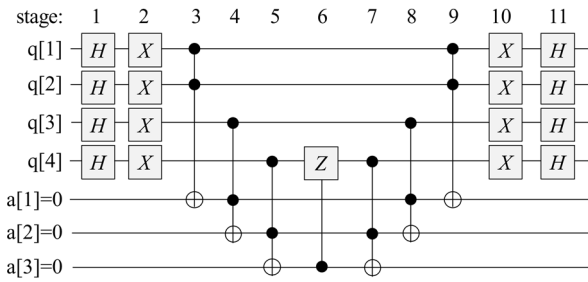


FIGURE 7. Realization of R_n for $n = 4$ based on Table 4.

processes at the beginning (stage 1) and the end (stage 11) represent the application of $H^{\otimes n}$, and the intermediate stages (stages 2 through 10) implement the operation $-D$. Within the $-D$ segment, stage 2 transforms the all-zero state in the data qubits into an all-one state. This sets off stages 3 to 5, which utilize the ancilla qubits to mark the last ancilla qubit a [3]. Stage 6 then utilizes this marking to flip the sign of the data qubits. Stages 7 to 9 reset the ancilla qubits to zero, preparing them for reuse in the algorithm's next iterations. Similarly, stage 10 switches the data qubits from the all-one state back to the all-zero state, ensuring that they are ready for further iterations of the algorithm.

Our proposed approach presupposes that n is greater than 1 (line 05). We do not address scenarios where only a single data qubit is present, corresponding to a case with $N = 2$ items. The rationale behind this exclusion is that Grover's algorithm loses its efficacy in such a minimal search space. When $N = 2$, the oracle's action of sign-inverting one of the two possible states results in them having equal but oppositely signed amplitudes. Consequently, the average amplitude of these states becomes zero. As a result, step 3, reflection around the mean, fails to alter the relative probabilities of the states, merely leading to a sign inversion of both. This renders Grover's algorithm ineffectual for $N = 2$.

To assess the effectiveness of the synthesized outcomes, we employ three metrics that align with those used in prior research [9], [22]: (i) the number of stages, (ii) the number of operators, and (iii) the number of ancilla qubits required. The rationale behind these metrics is as follows: minimizing the number of stages and operators can lead to reductions in both computational time and potential errors, while the number of qubits is a critical consideration given their status as a scarce resource on many quantum platforms. For the second metric, we count each n -qubit operator as contributing n towards the total count. For example, a sequence containing a single-qubit H operator, a two-qubit CZ operator, and a three-qubit $CCNOT$ operator would tally up to $1 + 2 + 3 = 6$ operators, reflecting the increased resource demand of larger n -qubit operators.

Table 5 presents a summary of these metrics. The method we propose requires two stages of $H^{\otimes n}$, two stages involving X operators, $2(n - 1)$ stages of $CCNOT$ operators, and one stage of a CZ operator, adding up to a total of

TABLE 5. Quality metrics of synthesis outcomes.

Measure	H	X	$CCNOT$	CZ	Sum
# of Stages	2	2	$2(n - 1)$	1	$2n + 3$
# of Operators	$2n$	$2n$	$2(n - 1) \times 3$	2	$10n - 4$
# of Ancilla Qubits	0	0	$n - 1$ (across $CCNOT$'s and CZ)		$n - 1$

n is the number of qubits needed to represent N items, calculated as $n = \lceil \log_2(N) \rceil$

$2n + 3$ stages. Each stage involving H and X operators consists of n operators, while each $CCNOT$ stage includes a single 3-qubit operator, and the CZ stage incorporates a 2-qubit operator, cumulating to $10n - 4$ operators in total. Additionally, $n - 1$ ancilla qubits are employed across the $CCNOT$ and CZ stages. In summary, all three of these evaluative metrics scale linearly with the number of qubits, n . When considering the number of items, N , these metrics scale with the logarithm base 2 of N , given that $n = \lceil \log_2(N) \rceil$.

IV. EVALUATION

We implemented and demonstrated the proposed methods, as outlined in Section III. In Section IV-A, we confirmed that our implementation effectively recognizes target items across a range of qubit numbers, n . In Section IV-B, we assessed the resource efficiency of our methods, relative to those in previous studies.

A. FUNCTIONAL VERIFICATION OF THE PROPOSED IMPLEMENTATION

Leveraging the techniques introduced in Section III, we implemented and executed Grover's algorithm for a spectrum of qubit counts, n . Subsequently, we conducted measurements on the data qubits to evaluate the frequency with which they accurately identified the target items. Our implementations were crafted using the Qiskit library [29] and executed on IBM's quantum computing platforms. In particular, we used the AerSimulator backend [30].

We implemented the three steps of Grover's algorithm (Fig. 1), in the following manner. Having already detailed the implementation of step 3 in Section III-B, we now elucidate the implementation of steps 1 and 2. For step 1, the initialization, we assigned all data qubits the state zero and then applied an H operator to each, generating an equal superposition across the data qubits. This prepares the ground for steps 2 and 3, which are responsible for amplifying the target item's probability. We also arranged n ancilla qubits in state zero. Specifically for the final ancilla qubit, $a[n]$, we induced the state $(1/\sqrt{2}, -1/\sqrt{2})$ by first applying an X operator and then an H operator. When step 2 identifies the

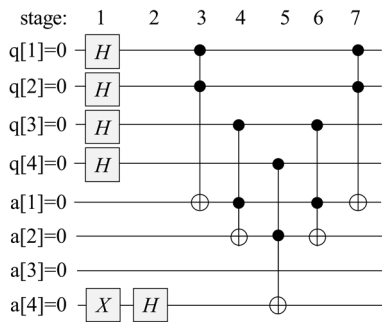


FIGURE 8. Realization of step 1 and step 2 for $n = 4$.

target item, it alters the state to $(-1/\sqrt{2}, 1/\sqrt{2})$, effectively sign-inverting the target item.

It might raise questions as to how the sign inversion of an ancilla qubit, $a[n]$, results in the sign inversion of the target state among the data qubits. The data qubits, placed in superposition, process multiple items simultaneously. Within this set, the ancilla qubit $a[n]$ undergoes sign inversion solely for the target item, while its state remains unchanged for all other items. This interaction forges a unique link between the sign-inverted $a[n]$ and the target item—a phenomenon known as entanglement [1], [24]. Entanglement implies that any operation affecting one part of an entangled pair impacts the whole system. Therefore, flipping the sign of the ancilla qubit $a[n]$ is reflected across the entangled pair, effectively flipping the sign of the target state within the data qubits.

We implemented step 2 of Grover's algorithm under the assumption that the target item is the all-one state. Should the target contain any zeros in specific qubits $q[i]$, we can initially perform X operations on these $q[i]$ to convert the target into an all-one state, and then proceed as detailed next. We located the target using a series of $CCNOT$ operators. These operators enable us to discern the all-one target state and execute an X operation on the last ancilla qubit, $a[n]$. Given the initial state of $a[n]$ set during step 1, the X operation transforms it to $(-1/\sqrt{2}, 1/\sqrt{2})$, corresponding to a sign inversion. Following this, a sequence of conditional operations is executed in reverse, which resets the ancilla qubits [1] through $a[n-1]$ to their original zero state. This reset ensures that the ancilla qubits are ready for reuse in the subsequent application of step 3.

Fig. 8 illustrates the implementation of steps 1 and 2 for $n = 4$. The initial stages 1 and 2 correspond to step 1 of Grover's algorithm, initializing both data and ancilla qubits. Subsequent stages, 3 to 7, align with step 2, the oracle. Among them, stages 3 to 5 detect the all-one state in data qubits via $CCNOT$ operators and invert the target item's sign by applying an X operation to the last ancilla qubit, $a[n]$. Stages 6 and 7 then reverse this sequence of conditional operations, resetting the ancilla qubits [1] to $a[n-1]$ to their initial zero state. Finally, the stages precede the implementation of step 3, as depicted in Fig. 7.

TABLE 6. Results of functional verification for various qubit counts.

Number of Data Qubits (n)	Total Items Represented (N)	Iterations of Steps 2 and 3 ($\lfloor \pi\sqrt{N}/4 \rfloor$)	Success Rate of Target Identification (%)
2	4	1	100.0%
3	8	2	93.7%
4	16	3	95.3%
5	32	4	100.0%
6	64	6	99.6%
7	128	8	99.6%
8	256	12	100.0%
9	512	17	99.8%
10	1,024	25	99.9%

Since our paper primarily focuses on the implementation of step 3 of Grover's algorithm, we do not extensively explore the specifics of configuring step 2, particularly in terms of mapping a given search space of size N to n qubits. For our purposes, we operated under the assumption that the target is encoded as the state with all qubits set to one, and accordingly, we tailored step 2 to invert the sign of this specific target state. However, in practical applications, the configuration of step 2 varies depending on the particular search problem at hand. Each problem has its own set of solution requirements and, therefore, requires different verification processes. In future research, we aim to delve into the nuances of implementing step 2, with the goal of developing and sharing guidelines that are broadly applicable to a variety of search objectives.

We executed the implementations for a range of qubit counts, n , and the results are compiled in Table 6. For each value of n , we performed steps 2 and 3 $\lfloor \pi\sqrt{N}/4 \rfloor$ times [5] followed by a measurement of the data qubits to capture a single outcome. This process was repeated a thousand times to calculate the success rate at which the implementations accurately determined the target item. Our results indicate a high success rate, with the target item being correctly identified in over 98.65% of the trials. This confirms the correctness of the proposed implementation methods. While there was a failure rate of 1.35%, additional replications and measurements would likely yield the correct target identification.

The number of iterations for steps 2 and 3 (i.e., $\lfloor \pi\sqrt{N}/4 \rfloor$) assumes the presence of a single target in the search space. When multiple targets exist, adjusting the number of iterations to $\lfloor \pi\sqrt{N}/M/4 \rfloor$ in the implementation can yield one of the targets with a probability exceeding 90% [25]. In this equation, M represents the number of targets, which can be estimated using a quantum counting algorithm as demonstrated in [31]. In this context, subsequent studies [32], [33] propose algorithms that can identify one of multiple targets without needing to pre-estimate the number of targets. In our future work, we plan to integrate such algorithms into our implementation.

B. COMPARATIVE ANALYSIS OF RESOURCE EFFICIENCY

We evaluated the resource usage of our proposed implementations in comparison to those documented in prior research. In particular, we assessed four metrics: the number of stages, operators, ancilla qubits, and the synthesis time required for the implementations. The criteria for the first three metrics are detailed in Section III-B. The synthesis time was determined by tracking how long it took to synthesize step 3 of Grover's algorithm, with any synthesis exceeding three hours labeled as unsuccessful.

Our methods were benchmarked against three existing approaches: (i) rule-based techniques, (ii) search-based techniques, and (iii) manual implementations found in the literature. For each of the categories, we selected one or two representative works, as the evaluation metrics exhibited similar trends across approaches within the same category.

In the first category, (i) rule-based techniques, we chose Qiskit's synthesis functionality [34], because it is fast, actively under development, and the code is publicly accessible. Qiskit breaks down a unitary matrix into smaller 1 or 2 qubit unitaries using a predefined set of rules [12]. Then it implements the 1 to 2 qubit unitaries using a combination of stochastic search and operator merging for efficient operator use [35].

In the second category, (ii) search-based techniques, we selected our previous work [18] that utilizes machine learning, because it is recent and produces either more compact implementations compared to other approaches or comparably compact ones. For example, it implements the 2-qubit Grover's iterator with six operators, as illustrated in Fig. 3(b), which is fewer than other known implementations. In terms of synthesis time, we observed that most approaches in this category encounter difficulties when producing implementations for qubit counts greater than 5. The chosen machine-learning-based approach [18] learns from a variety of quantum implementations to identify a compact set of operators for efficient unitary matrix implementation. Consequently, it discovers compact implementations and reduces search time compared to using a full operator set.

In addition to our previous work, we also compared our proposed method with QSyn [21], one of the most recent synthesis frameworks in the second category, where codes are publicly available. QSyn enumerates various implementations by assembling modules of quantum operators, rather than individual operators. Since a module represents a frequently-reused arrangement of operators, synthesizing at the module level can expedite the process compared to assembly at the operator level.

In the third category, (iii) manual implementations, we selected manually optimized implementations found in the existing literatures [5] and [6]. For example, Fig. 3(a) illustrates the implementation for $n = 2$ qubits. Most other implementations in the literature do not significantly differ in terms of stage numbers and operator counts.

For our experimentation, we developed our proposed methods using Python and applied them to synthesize step 3 of Grover's algorithm across different qubit counts, n . In particular, we built a synthesizer capable of generating Python scripts that execute quantum searches via the Qiskit library [29]. We conducted the tests on a system equipped with a 3.4GHz Intel Core i7-6700 processor and 16GB of RAM.

Table 7 presents a summary of our experimental findings. The rule-based technique quickly produced implementations for qubit counts from 2 to 6. However, we observed an exponential increase in both the number of stages and operators as the qubit count rose. On the other hand, the search-based technique yielded implementations that required significantly fewer stages and operators but took a considerably longer time for synthesis. The rationale behind this is that the rule-based technique consistently adds operators in a set pattern to incrementally approach the desired unitary matrix, whereas the search-based explores a more diverse space of potential implementations to find a compact one. Regarding manual implementations, we identified examples for $n = 2$ and 3. In these cases, we observed that they utilized marginally higher numbers of stages and operators compared to the implementations derived from the search-based techniques.

Our developed methods strike a balance between rule-based and search-based approaches. While our implementations slightly exceed the search-based results and manual implementations in terms of the number of stages, operators, and the use of additional ancilla qubits, the rate at which these numbers increase is significantly less than that of the rule-based technique, showing a linear growth with the increase in qubits. Furthermore, the synthesis time for our proposed methods generally fell below that required by the other techniques. The key differences observed stem largely from the fact that rule-based and search-based techniques are designed to handle a broad range of unitary matrices, which might not always yield the most efficient results. In contrast, our approach is specifically tailored to optimize the implementation of Grover's algorithm.

We conducted experiments up to a qubit count of 10. This was due to the rule-based method exhibiting a steep increase in the number of stages and operators as the qubit count grows. Similarly, the search-based method displayed a significant increase in synthesis time, failing to synthesize implementations for systems with 5 or more qubits. However, our proposed method is capable of producing implementations for a larger number of qubits, provided that the underlying quantum hardware supports the required number of qubits.

We now discuss the implications of using ancilla qubits in our proposed method. Unlike other methods, our approach requires $n - 1$ ancilla qubits (as shown in Table 5). While current quantum hardware has a limited number of qubits, we anticipate that the use of ancilla qubits will impose a moderate burden in the long term for the following reasons.

TABLE 7. Resource usage of various synthesis techniques.

Rule-based Technique (Qiskit) [34]				
# of Data Qubits (n)	# of Stages	# of Operators	# of Ancilla Qubits	Time to Synthesize
2	8	14	0	0.37s
3	22	40	0	0.98s
4	34	69	0	2.16s
5	65	125	0	2.98s
6	128	229	0	6.83s
7	255	429	0	36.08s
8	508	816	0	61.14s
9	1015	1648	0	282.43s
10	2029	3279	0	1,230.78s
Search-based Technique 1 [18]				
# of Data Qubits (n)	# of Stages	# of Operators	# of Ancilla Qubits	Time to Synthesize
2	3	6	0	0.35s
3	5	12	0	334.55s
4	7	20	0	1,083.18s
5–10	Fail (i.e., synthesis time > 3hrs)			
Search-based Technique 2 (QSyn) [21]				
# of Data Qubits (n)	# of Stages	# of Operators	# of Ancilla Qubits	Time to Synthesize
2	4	6	0	13.44s
3–10	Fail (i.e., synthesis time > 3hrs)			
Manual Implementation [5,6]				
# of Data Qubits (n)	# of Stages	# of Operators	# of Ancilla Qubits	Time to Synthesize
2	7	12	0	N/A
3	7	17	0	N/A
4–10	Not found in the literature			
Proposed Methods				
# of Data Qubits (n)	# of Stages	# of Operators	# of Ancilla Qubits	Time to Synthesize
2	7	16	1	0.05s
3	9	26	2	0.10s
4	11	36	3	0.13s
5	13	46	4	0.17s
6	15	56	5	0.21s
7	17	66	6	0.31s
8	19	76	7	0.37s
9	21	86	8	0.42s
10	23	96	9	0.48s
50	103	496	49	1.58s
100	203	996	99	2.73s

Firstly, $n = \lceil \log_2(N) \rceil$, where N is the size of the search space. Hence, the number of required ancilla qubits grows logarithmically as the search space increases. Secondly, our proposed implementation ensures that all ancilla qubits are reset to their initial values after each application of step 3. As a result, the same ancilla qubits can be reused for subsequent applications of step 2 (the oracle) as needed. In summary, our method utilizes a moderate number of ancilla qubits, but the advantages of using them (i.e., savings in synthesis time and the compactness of the resulting implementations) outweigh the cost.

V. CONCLUSION

Our research introduces techniques for synthesizing a key segment of Grover's quantum search algorithm. This segment is crucial for amplifying the amplitude of the search target while reducing those of non-target items, making the target more distinguishable. This part of the algorithm is universal and does not vary with different search objectives. Our methods are designed to work for a range of search space sizes, represented by the number of qubits in the algorithm. We developed and evaluated these methods, showing that they can efficiently produce implementations with a moderate and linearly increasing number of stages and operators as the qubit count grows. We anticipate that the proposed methods facilitate large-scale implementations of Grover's algorithm across various domains, including database searches, cryptanalysis, pattern recognition, quantum simulations, and more [5].

Moving forward, our focus will shift to developing techniques for another critical component of Grover's algorithm, known as the oracle. This element distinguishes the search target from others by inverting its sign, setting the stage for subsequent amplitude amplification. Since the oracle's design is inherently linked to specific search objectives, our aim is to create a versatile oracle applicable to various goals, establishing a set of broad guidelines for its implementation across different search scenarios.

REFERENCES

- [1] C. Bernhardt, *Quantum Computing for Everyone*. Cambridge, MA, USA: MIT Press, 2020.
- [2] H. Deng, Y. Peng, M. Hicks, and X. Wu, "Automating NISQ application design with meta quantum circuits with constraints (MQCC)," *ACM Trans. Quantum Comput.*, vol. 4, no. 3, pp. 1–29, Apr. 2023, doi: 10.1145/3579369.
- [3] J. Vos, *Quantum Computing in Action*. Shelter Island, NY, USA: Manning, 2022.
- [4] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proc. 28th Annu. ACM Symp. Theory Comput.*, Philadelphia, PA, USA, 1996, pp. 212–219, doi: 10.1145/237814.237866.
- [5] A. J. et al., "Quantum algorithm implementations for beginners," *ACM Trans. Quantum Comput.*, vol. 3, no. 4, pp. 1–92, Jul. 2022, doi: 10.1145/3517340.
- [6] R. Loredo, *Learn Quantum Computing With Python and IBM Quantum Experience*. Birmingham, U.K.: Packt, 2020.
- [7] T. Atkinson, A. Karsa, J. Drake, and J. Swan, "Quantum program synthesis: Swarm algorithms and benchmarks," in *Proc. Eur. Conf. Genetic Program.*, Leipzig, Germany, 2019, pp. 19–34, doi: 10.1007/978-3-030-16670-0_2.
- [8] A. Zulehner and R. Wille, *Introducing Design Automation for Quantum Computing*. Berlin, Germany: Springer, 2020.
- [9] M. Amy, D. Maslov, M. Mosca, and M. Roetteler, "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 6, pp. 818–830, Jun. 2013, doi: 10.1109/TCAD.2013.2244643.
- [10] A. Rajaei, M. Houshmand, and S. A. Hosseini, "A dynamic programming approach to multi-objective logic synthesis of quantum circuits," *Quantum Inf. Process.*, vol. 22, no. 10, p. 384, Oct. 2023, doi: 10.1007/s1128-023-04112-z.
- [11] A. M. Krol, A. Sarkar, I. Ashraf, Z. Al-Ars, and K. Bertels, "Efficient decomposition of unitary matrices in quantum circuit compilers," *Appl. Sci.*, vol. 12, no. 2, p. 759, Jan. 2022, doi: 10.3390/app12020759.
- [12] R. Iten, O. Reardon-Smith, E. Malvetti, L. Mondada, G. Pauvert, E. Redmond, R. S. Kohli, and R. Colbeck, "Introduction to UniversalQ-Compiler," 2019, *arXiv:1904.01072*.

- [13] V. V. Shende, S. S. Bullock, and I. L. Markov, "Synthesis of quantum-logic circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 6, pp. 1000–1010, Jun. 2006, doi: [10.1109/TCAD.2005.855930](https://doi.org/10.1109/TCAD.2005.855930).
- [14] M. Saeedi, M. Arabzadeh, M. S. Zamani, and M. Sedighi, "Block-based quantum-logic synthesis," *Quantum Inf. Comput.*, vol. 11, no. 3, pp. 262–277, Mar. 2011, doi: [10.26421/qic11.3-4-6](https://doi.org/10.26421/qic11.3-4-6).
- [15] P. Niemann, R. Wille, and R. Drechsler, "Efficient synthesis of quantum circuits implementing Clifford group operations," in *Proc. 19th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Suntec, Singapore, Jan. 2014, pp. 483–488, doi: [10.1109/ASPAC.2014.6742938](https://doi.org/10.1109/ASPAC.2014.6742938).
- [16] P. Niemann, R. Willie, and R. Drechsler, "Improved synthesis of Clifford+T quantum functionality," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Dresden, Germany, Mar. 2018, pp. 597–600, doi: [10.23919/DATE.2018.8342078](https://doi.org/10.23919/DATE.2018.8342078).
- [17] J. J. Vartiainen, M. Möttönen, and M. M. Salmaa, "Efficient decomposition of quantum gates," *Phys. Rev. Lett.*, vol. 92, no. 17, Apr. 2004, Art. no. 177902, doi: [10.1103/physrevlett.92.177902](https://doi.org/10.1103/physrevlett.92.177902).
- [18] S. Lee and S. Y. Nam, "Quantum program synthesis through operator learning and selection," *IEEE Access*, vol. 11, pp. 25755–25767, 2023, doi: [10.1109/ACCESS.2023.3257192](https://doi.org/10.1109/ACCESS.2023.3257192).
- [19] Y. Xiao, S. Nazarian, and P. Bogdan, "A stochastic quantum program synthesis framework based on Bayesian optimization," *Sci. Rep.*, vol. 11, no. 1, p. 13138, Jun. 2021, doi: [10.1038/s41598-021-91035-3](https://doi.org/10.1038/s41598-021-91035-3).
- [20] G. Alvarez, R. Bennink, S. Irlle, and J. Jakowski, "Gene expression programming for quantum computing," *ACM Trans. Quantum Comput.*, vol. 4, no. 4, pp. 1–14, Oct. 2023, doi: [10.1145/3617691](https://doi.org/10.1145/3617691).
- [21] C. G. Kang and H. Oh, "Modular component-based quantum circuit synthesis," *Proc. ACM Program. Lang.*, vol. 7, pp. 348–375, Apr. 2023, doi: [10.1145/3586039](https://doi.org/10.1145/3586039).
- [22] E. Younis, K. Sen, K. Yelick, and C. Iancu, "QFAST: Conflating search and numerical optimization for scalable quantum circuit synthesis," in *Proc. IEEE Int. Conf. Quantum Comput. Eng. (QCE)*, Broomfield, CO, USA, Oct. 2021, pp. 232–243, doi: [10.1109/QCE52317.2021.00041](https://doi.org/10.1109/QCE52317.2021.00041).
- [23] H. Deng, R. Tao, Y. Peng, and X. Wu, "A case for synthesis of recursive quantum unitary programs," *Proc. ACM Program. Lang.*, vol. 8, pp. 1759–1788, Jan. 2024, doi: [10.1145/3632901](https://doi.org/10.1145/3632901).
- [24] W. Scherer, *Mathematics of Quantum Computing*. Berlin, Germany: Springer, 2019.
- [25] M. Boyer, G. Brassard, P. Høyer, and A. Tapp, "Tight bounds on quantum searching," *Fortschritte der Physik*, vol. 46, nos. 4–5, pp. 493–505, Jun. 1998, doi: [10.1002/3527603093.ch10](https://doi.org/10.1002/3527603093.ch10).
- [26] S. Paramasivam, J. Jenitha, S. Sanjana, and M. Haghparast, "Compact quantum circuit design of PUFFIN and PRINT lightweight ciphers for quantum key recovery attack," *IEEE Access*, vol. 11, pp. 66767–66776, 2023, doi: [10.1109/ACCESS.2023.3289764](https://doi.org/10.1109/ACCESS.2023.3289764).
- [27] E. Johnston and N. Harrigan, *Programming Quantum Computers: Essential Algorithms and Code Samples*. Sebastopol, CA, USA: O'Reilly, 2019.
- [28] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge, U.K.: Cambridge Univ. Press, 2011.
- [29] *Qiskit: Open-Source Toolkit for Quantum Development*. Accessed: Nov. 20, 2023. [Online]. Available: <https://qiskit.org>
- [30] *AerSimulator*. Accessed: Nov. 20, 2023. [Online]. Available: https://qiskit.org/ecosystem/aer/stubs/qiskit_aer.AerSimulator.html
- [31] G. Brassard, P. Hoyer, and A. Tapp, "Quantum counting," in *Proc. Int. Colloq. Automata, Lang., Program.*, in Lecture Notes in Computer Science, vol. 1443, Jan. 2006, pp. 820–831, doi: [10.1007/BFb0055105](https://doi.org/10.1007/BFb0055105).
- [32] B. Yan, S. Wei, H. Jiang, H. Wang, Q. Duan, Z. Ma, and G.-L. Long, "Fixed-point oblivious quantum amplitude-amplification algorithm," *Sci. Rep.*, vol. 12, no. 1, p. 14339, Aug. 2022, doi: [10.1038/s41598-022-15093-x](https://doi.org/10.1038/s41598-022-15093-x).
- [33] L. K. Grover, "Fixed-point quantum search," *Phys. Rev. Lett.*, vol. 95, no. 15, pp. 150501–150504, Oct. 2005, doi: [10.1103/physrevlett.95.150501](https://doi.org/10.1103/physrevlett.95.150501).
- [34] *Qiskit Transpiler*. Accessed: Nov. 20, 2023. [Online]. Available: <https://qiskit.org/documentation/apidoc/transpiler.html>
- [35] S. Bravyi, R. Shaydulin, S. Hu, and D. Maslov, "Clifford circuit optimization with templates and symbolic Pauli gates," *Quantum*, vol. 5, no. 1, p. 580, Nov. 2021, doi: [10.22331/q-2021-11-16-580](https://doi.org/10.22331/q-2021-11-16-580).



SIHYUNG LEE received the B.S. (summa cum laude) and M.S. degrees in electrical engineering from Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2000 and 2004, respectively, and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University (CMU), USA, in 2010. From 2010 to 2011, he was a Postdoctoral Researcher with the Network Management Group, IBM Thomas J. Watson Research Center, USA.

From 2011 to 2019, he was a Professor with the Department of Information Security, Seoul Women's University, South Korea. Since 2019, he has been a Professor with the School of Computer Science and Engineering, Kyungpook National University. His research interests include pattern mining from social network traffic and program synthesis for classical and quantum computers.



SEUNG YEOB NAM (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 1997, 1999, and 2004, respectively. From 2004 to 2006, he was a Postdoctoral Research Fellow with the CyLab, Carnegie Mellon University. In 2007, he joined the Department of Information and Communication Engineering, Yeungnam University, Gyeongsan, South Korea,

where he is currently a Professor. From January 2012 to January 2013, he was a Visiting Professor with the Department of Electrical and Computer Engineering, Carnegie Mellon University. His research interests include network security, blockchain, network management, and wireless networks. He received the 2022 Best Paper Award from Digital Communications and Networks (DCN).

...