THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

SCHOOL OF ENGINEERING

# APPLYING RECURSIVE QUANTUM CIRCUIT GENERATION TO SOLVE DATABASE SEARCH PROBLEMS USING GROVER'S ALGORITHM

NATHANIEL V. RONDINELLI
SPRING 2024

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Software Engineering
with honors in Software Engineering

Reviewed and approved* by the following:

Dr. Wen-Li Wang
Professor of Software Engineering
Thesis Supervisor and Honors Advisor

Dr. Shahid Hussain
Professor of Software Engineering
Thesis Reader

* Electronic approvals are on file.

## Abstract

**Context:** Grover's algorithm is a powerful tool in the realm of quantum computing because it offers exponential speedup over classical linear searching for unstructured search problems. However, it is challenging to construct efficient and scalable circuits for Grover's algorithm, particularly with increasing numbers of qubit inputs. **Problem:** Traditional circuit construction methods often require manual composition and result in lengthy and complex circuits. This hinders practical implementation, and the difficulty increases as the search space grows exponentially with additional qubits. **Method:** The proposed research provides the approach of recursively generating oracle circuits using Toffoli gates as the fundamental units to enhance scalability and reusability while maintaining effectiveness. The methodology leverages Qiskit and IBM Quantum Lab simulation for computation due to limited access to quantum hardware. **Result:** Through experiments and comparisons between recursively and non-recursively generated circuits, the findings suggest that the recursive approach offers comparable accuracy and computational complexity to the non-recursive approach. However, it tends to require a larger number of basic quantum gates. **Conclusion:** This research highlights the possibility of utilizing recursively generated circuits as an alternative when implementing Grover's algorithm. It presents potential scalability benefits and creates an avenue for further research into the time complexity of using actual quantum hardware to validate these results.

## Table of Contents

## List of Figures

## List of Tables

## Acknowledgements

I would like to acknowledge and thank my advisor, Dr. Wen-Li Wang, for his guidance and direction throughout this research project. His advice and expertise were critical throughout this process of learning and overcoming challenges. I also want to acknowledge and thank my co-advisor, Dr. Shahid Hussain, for his time and input in reviewing this paper and his ability and willingness to act as an additional advisor.

# Chapter 1 Introduction

## 1.1 Background

In the field of computing, database search problems have been studied to minimize the amount of time required to find a specific record within a database. Many classical solutions have been proposed over the years, eventually culminating in the binary search. Binary searching serves as the optimal way to search a sorted database, while linear searching is the most suitable method for searching unsorted data. However, quantum computing provides a solution in Grover's algorithm [11] which takes advantage of quantum speedup to search unsorted databases faster.

Quantum computing deviates greatly from classical computing because it relies upon the principles of quantum physics to perform operations. Classical bits represent a binary state of either 0 or 1, but quantum bits (qubits) utilize quantum properties such as superposition and entanglement to exist in several states simultaneously. Because of these phenomena, quantum computers possess the unique ability to explore a large solution space at incredible speed, thus enabling greater processing power for specific information processing tasks. At a hardware architecture level, quantum circuits are composed of quantum gates. These gates can be understood as functions that alter qubits' states and respective probabilities. A quantum gate takes in an initial set of qubits and then outputs a distinctly transformed set of qubits with modified state and probability. Quantum algorithms refer to algorithms that take advantage of the inherent principles of quantum mechanics to achieve higher efficiency when compared to classical algorithms [1]. These algorithms perform certain functions by utilizing a combinational arrangement of quantum gates as components, which must be completely reversible.

### 1.2 Objective

This proposed research aims to produce a practical application of recursively generated quantum circuits. It will be tasked in solving unstructured database search problems using Grover's algorithm with varying numbers of qubits. These circuits should consist of Toffoli gates as the primary building block and will be based on the work of Barenco et al. [6]. Comparisons will be made across the metrics defined in Chapter 5 to determine the viability of using Grover's algorithm with recursively and non-recursively generated oracle circuits.

### 1.3 Significance

The significance of this proposed research relates to hardware development and to solving the unsorted database search problem with an n-qubit implementation of Grover's algorithm. For every additional input qubit, the possible search space increases exponentially. If this recursive approach to constructing quantum circuits could be realized and optimized, then it could vastly increase the effectiveness of using quantum computing for database search operations and other problem domains.

### 1.4 Outline

Chapter 2 presents a literature review of the related research conducted by other professionals in the realm of quantum computing. This includes studies related to quantum gates, algorithms, and qubits.

Chapter 3 describes in detail the problem of circuit construction and the difficulty of achieving gates with many qubit inputs.

Chapter 4 discusses the technologies and methods used to collect data and proposes solutions to the given problem.

Chapter 5 showcases and analyzes the results of the experiments and then makes comparisons across metrics such as the number of fundamental gates used in circuit construction.

Chapter 6 draws conclusions from the data regarding performance and optimization, describes the challenges faced, and provides recommendations for future research.

# Chapter 2 Related Works

## 2.1 Qubits and Data

Like a classical bit, a quantum bit (qubit) serves as the base unit of quantum information. The advantage of using qubits stems from quantum phenomena such as superposition, entanglement, and parallelism [2]. For example, the property which allows qubits to represent a probabilistic combination of states is called superposition. Rather than representing exclusively 0 or 1 as classical bits do, qubits can simultaneously exist in several states [1] because they maintain a probability of being either 0 or 1 at any given time of measurement. The sum of these probabilities is always equal to 1. Quantum entanglement, however, is when the state of one qubit becomes related (entangled) with the state of another qubit. Because this is independent of physical distance, it allows for so-called 'quantum teleportation' of data. For example: one qubit can transmit data instantaneously to another entangled qubit, regardless of physical separation [5]. Parallelism refers to quantum systems performing computations on multiple qubits simultaneously thus contributing greatly to quantum speedup. Another comparison between quantum bits and classical bits is that of data representation.

The number of qubits utilized within a given circuit directly corresponds to how much data can be represented by the circuit. For example, both a classical circuit of three bits and a quantum circuit of three qubits can each represent $2^3 = 8$ number of states. At first glance, this may appear like they are equal. However, the quantum system proves more powerful due to its utilization of unique properties that allow it to perform computations more efficiently and explore solution spaces in parallel. Increasing from three qubits to four qubits results in a non-linear increase in

the data representation such that every additional qubit increases the state space by twofold. This exponential increase naturally gains importance and relevance when faced with problems involving large solution spaces such as the unstructured search problem.

**2.2 Quantum Gates**

In general, two primary quantum computing paradigms exist: the annealing-based paradigm and the gate-model paradigm. Annealing-based quantum computers perform a process called quantum annealing, which returns the minimum energy state of a quantum energy function known as a Hamiltonian. In the case of optimization problems, the quantum Hamiltonian is defined according to the fitness function of the given problem such that the solution is represented by the lower energy state [3]. Quantum-gate devices, on the other hand, utilize operations called quantum gates, which are comparable to the classical logic gates of traditional circuitry. Quantum gates are sequentially applied to qubit states, transforming the states until a solution is reached [3].

Basic gates include 1-qubit and 2-qubit quantum gates. These titles refer to the number of qubits that are utilized or acted upon by the gate's operation, which may increase up to any number of qubits. Fundamental quantum gates exist such as the Hadamard gate, phase shift gate, X gate, CX gate, and Toffoli gate [4]. The Hadamard gate is a single-qubit gate which employs the matrix $H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$. It functions to map the qubit-basis states $\{ |0\rangle, |1\rangle \}$ (also called computational basis states) into superposition states $\left\{\frac{|0\rangle + |1\rangle}{\sqrt{2}}, \frac{|0\rangle - |1\rangle}{\sqrt{2}}\right\}$ (also called polar basis states) respectively [2]. Another fundamental gate is a phase-shift gate, also known as the Z gate, which rotates $\pi$ radians about the Bloch sphere's z axis [4]. The Z gate is a single-qubit gate that functions to map the basis state $|0\rangle$ into $|0\rangle$ and the basis state $|1\rangle$ into $e^{i\pi}|1\rangle$. This can be represented by the matrix

$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$. This phase-flip operation simply alters the geometrical orientation of the qubit without impacting the probability of measuring $|0\rangle$ or $|1\rangle$. The X gate, CX gate, and Toffoli gate share similarity with one another. The quantum X gate is a single-qubit gate that functions like the classical NOT gate, applying a bit-flip operation to the input. Therefore, it maps the basis state $|0\rangle$ into $|1\rangle$ and $|1\rangle$ into $|0\rangle$. Its matrix representation is $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. The CX gate is a two-qubit gate which uses the first qubit (q_1) as a control and the second qubit (q_0) as the result qubit as shown below in *Figure 1*:

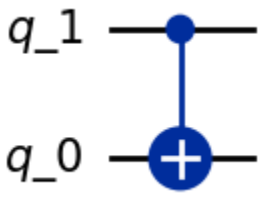


**Figure 1. CX Gate with Qubit Labels**

*Figure 1* demonstrates an example of the CX gate with qubit labels. These qubits are referenced using the notation, $|q_1 q_0\rangle$, which represents the composite result of $|q_1\rangle|q_0\rangle$. Like the X gate, the CX gate also performs a bit-flip on the result qubit. However, the control qubit is responsible for controlling the activation of the gate's function such that the bit-flip operation occurs if and only if the control qubit $q_1$ has state $|1\rangle$, and the initial state of result qubit $q_0$ maintains no control [4]. The CX gate maps input state $|00\rangle$ into $|00\rangle$, state $|01\rangle$ into $|01\rangle$, state $|10\rangle$ into $|11\rangle$, and state $|11\rangle$ into $|10\rangle$. Its matrix representation is given as $CX = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$.

Toffoli is the proper name given to the 2-control X gate. It functions like the CX gate, but it contains an extra control qubit that can be called $q_2$. It works in the same way such that a bit-flip of the result qubit $q_0$ occurs if and only if both control qubits $q_1$ and $q_2$ have state $|1\rangle$. The Toffoli gate is a 3-qubit quantum gate with matrix representation given below in *Equation 1*:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

**Equation 1. Matrix Representation of Toffoli Gate**

This results in the mapping of input states $|000\rangle$ into $|000\rangle$, $|001\rangle$ into $|001\rangle$, $|010\rangle$ into $|010\rangle$, $|011\rangle$ into $|011\rangle$, $|100\rangle$ into $|100\rangle$, $|101\rangle$ into $|101\rangle$, $|110\rangle$ into $|111\rangle$, and $|111\rangle$ into $|110\rangle$ [9]. Its visual representation is given below in *Figure 2*:

**Figure 2. Toffoli Gate**

The Toffoli gate, as depicted in *Figure 2*, was used as the fundamental building block for recursive circuit construction as described in Chapters 4 and 5. These fundamental gates hold critical importance for building circuits that quantum algorithms use to solve complex problems.

### 2.3 Quantum Algorithms

Quantum algorithms operate in a very similar fashion to classical algorithms in that both require input, perform computation, and produce output. However, each differs in terms of its specific hardware capabilities. Classical computers utilize binary bits and logic operations to perform computation, while quantum computers leverage qubits and properties of superposition, entanglement, and parallelism [5]. For this reason, classical results are considered deterministic and quantum results are considered probabilistic, and quantum machines must complete many iterations of a given algorithm to generate a statistically reliable result. Notable quantum algorithms are the Deutsch-Jozsa algorithm [10], Grover's Algorithm [11], and Shor's algorithm [12].

The Deutsch-Jozsa algorithm maintains importance because it demonstrated an early example of quantum speedup, that is, an exponential increase in the speed at which a solution is found. In this algorithm, the speedup is showcased using an oracle and a Boolean function. The Deutsch-Jozsa algorithm solves a black box problem wherein the oracle determines whether the output of the Boolean function is constant (either all 1 or all 0) or balanced (half 1 and half 0) [11]. For n inputs, the classical algorithm requires $2^{n-1} + 1$ iterations to determine the solution [5], but the quantum algorithm requires only 1 iteration to determine the solution. This demonstrates an exponential speedup because the time complexity is reduced by $2^n$.

Grover's algorithm is a well-known quantum algorithm leveraged for solving unstructured search problems [11]. It completes the task of locating a known element ω from an unordered database of size N. This element ω is the only input that can achieve output of 1 (true) from a Boolean function, while all other inputs result in 0 (false), thus making ω the solution. In classical computing, identifying ω from a list of N unsorted records requires O(N) time complexity in the worst case and O(N/2) on average. However, Grover's algorithm performs better when compared to the classical method by completing the task in $O(\sqrt{N})$ time complexity [1]. Grover's algorithm uses quantum parallelism to increase the search speed and will be described in greater detail in Chapter 3.

Shor's algorithm is a quantum algorithm primarily suited for cryptography applications. It reduces the time complexity of factoring large prime numbers from exponential to polynomial time because of speedup during the quantum Fourier transform (QTF) step [12]. The algorithm follows a three-step sequence: period finding, then quantum Fourier transform, and then integer factorization [5]. Shor's algorithm is held back by the fact that each QFT requires a custom quantum circuit, but it could become more powerful with the implementation of automatic circuit generation for the Fourier transform step. The common link between these quantum algorithms is that their effectiveness comes from leveraging the properties of quantum physics manifested in the form of qubits.

# Chapter 3 Problem Statement

### 3.1 Unstructured Search Problem

The problem which Grover's algorithm was made to solve is the unstructured search problem [11]. Recall that classically, the process of locating an element ω in an unsorted database of size N requires O(N) time complexity in the worst case and O(N/2) on average, but Grover's algorithm provides a quantum speedup resulting in O(√N) time complexity [1]. This element ω is the only input to a Boolean function f which outputs 1, and all other elements output 0. That is, $f(\omega) = 1$ and $f(x) = 0$ for all $x \neq \omega$. The algorithm uses an oracle circuit to perform the search computations while separate amplification circuit boosts the solution accuracy. The algorithm sequence is described below in *Figure 3*:

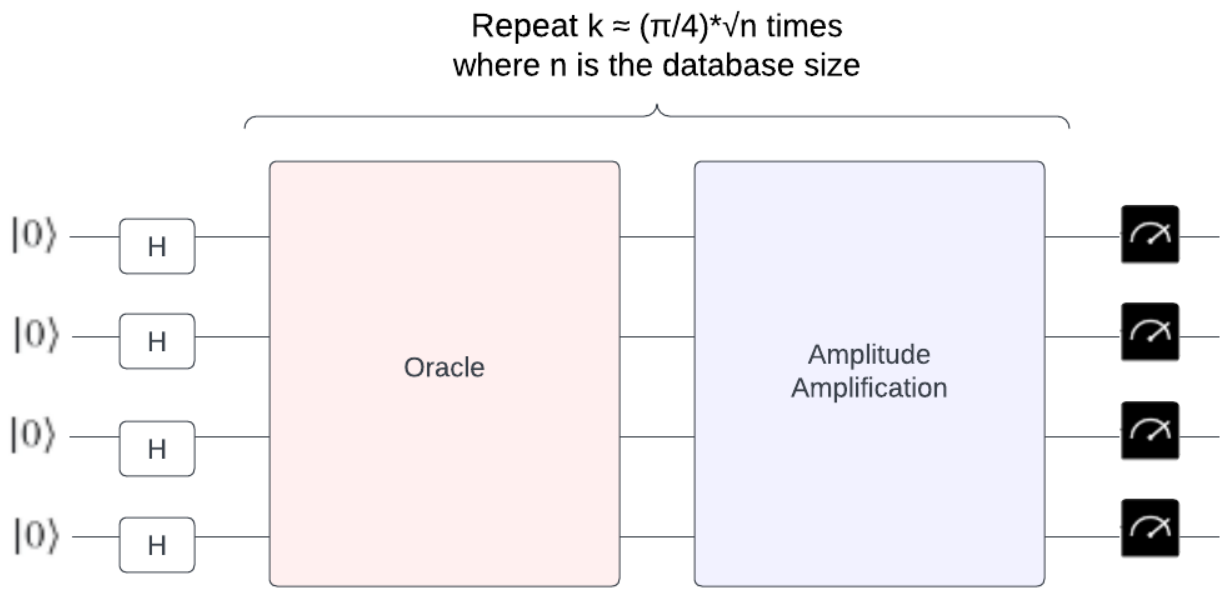


**Figure 3. Diagram of Grover's Algorithm**

The sequence of steps in Grover's algorithm described in *Figure 3* are as follows: 1.) Initialize all qubits to the |0⟩ state. 2.) Apply a Hadamard gate to each qubit to achieve uniform superposition. 3.) Apply both the oracle and amplification circuits $k \approx \frac{\pi}{4}\sqrt{N}$ number of times where N is the database size in number of records. 4.) Measure output [1] with respect to the standard basis: { |0⟩, |1⟩ }. Because this algorithm uses O(√N) number of computations instead of O(N) number of computations, it reduces the time complexity from linear to quadratic respectively.

Grover's algorithm begins with initializing each of m qubits with the starting state |0⟩, and then a Hadamard operation is applied to each qubit. This achieves a uniform superposition of $2^m$ basis states [1], denoted as ψ (psi), where m is the number of qubits. This occurs according to *Equation 2* as given below:

$$|\psi_0\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

**Equation 2. Uniform State Initialization**

The oracle and amplification circuits are then applied for $k \approx \frac{\pi}{4}\sqrt{N}$ number of iterations [1]. The oracle is responsible for performing a phase-flip on the amplitude of entry ω. Recall the Boolean function f which is satisfied by input ω such that $f(\omega) = 1$. The oracle O operates on the quantum state |ψ⟩ according to *Equation 3* as given below with $\alpha_x$ representing amplitude:

$$O|\psi\rangle = \sum_{x=0}^{N-1} (-1)^{f(x)} \alpha_x |x\rangle$$

**Equation 3. Grover's Oracle Operation**

Amplitude amplification [1] then occurs according to *Equation 4* as given below:

$$A = (2|\psi_0\rangle\langle\psi_0| - I)O$$

**Equation 4. Grover's Amplitude Amplification**

These steps are subsequently repeated, resulting in a probability of accurately locating ω after k iterations [1] given by *Equation 5* below where $k \approx \frac{\pi}{4}\sqrt{N}$ and $\theta = \sin^{-1}(\frac{1}{\sqrt{N}})$:

$$P(k) = |\langle w|\psi\rangle|^2 = \sin^2((2k+1)\theta)$$

**Equation 5. Grover's Result Probability**

These mathematical processes make up the computations that occur during runtime of Grover's algorithm and are responsible for producing the quantum speedup.

### 3.2 Circuit Construction

One challenge with using Grover's algorithm is that the necessary oracle circuit can be difficult to construct in the case of many qubit inputs. In an unstructured search for database size of 8 records (three qubits), there is not much benefit to using a quantum algorithm because the size of the database is so small. The benefits of Grover's algorithm can be utilized more effectively in the case of searching within a larger database. To construct an oracle with the ability to search twice as many records (16 rather than 8), it would require one additional qubit. This can theoretically be scaled up to facilitate database searching of 1,000 or even 1,000,000 records, but the difficulty comes in constructing circuits with such a high number of qubits. These circuits

become extremely long and are complex to design. For example, *Figure 4* shows a 3-control X gate and *Figure 5* shows that gate's decomposition.


**Figure 4. Quantum 3-control X gate**

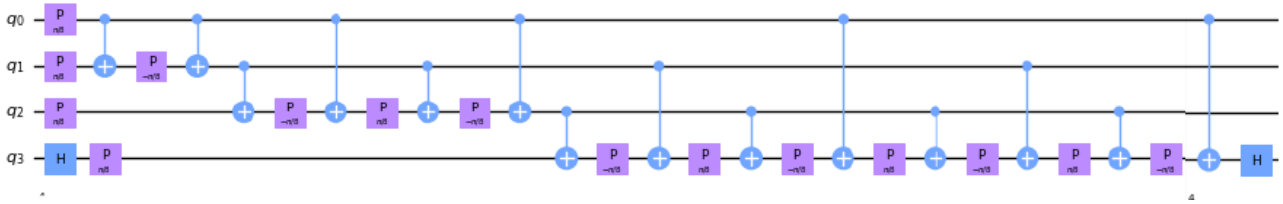


**Figure 5. Decomposition of Quantum 3-control X gate**

Circuit decomposition is an important metric as it describes the complexity of a circuit by showcasing the circuit's full set of internal gates. The circuit decomposition shown in *Figure 5* demonstrates how many 1-qubit and 2-qubit gates are required to construct the gate shown in *Figure 4*. The 3-control X gate is related to the Toffoli gate because it enhances the functionality of the Toffoli gate, due to containing one additional control qubit. However, the addition of control qubits causes the number of internal gates to grow, as showcased by *Figure 6* and *Figure 7* below:

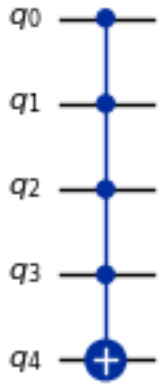


**Figure 6. Quantum 4-control X gate**

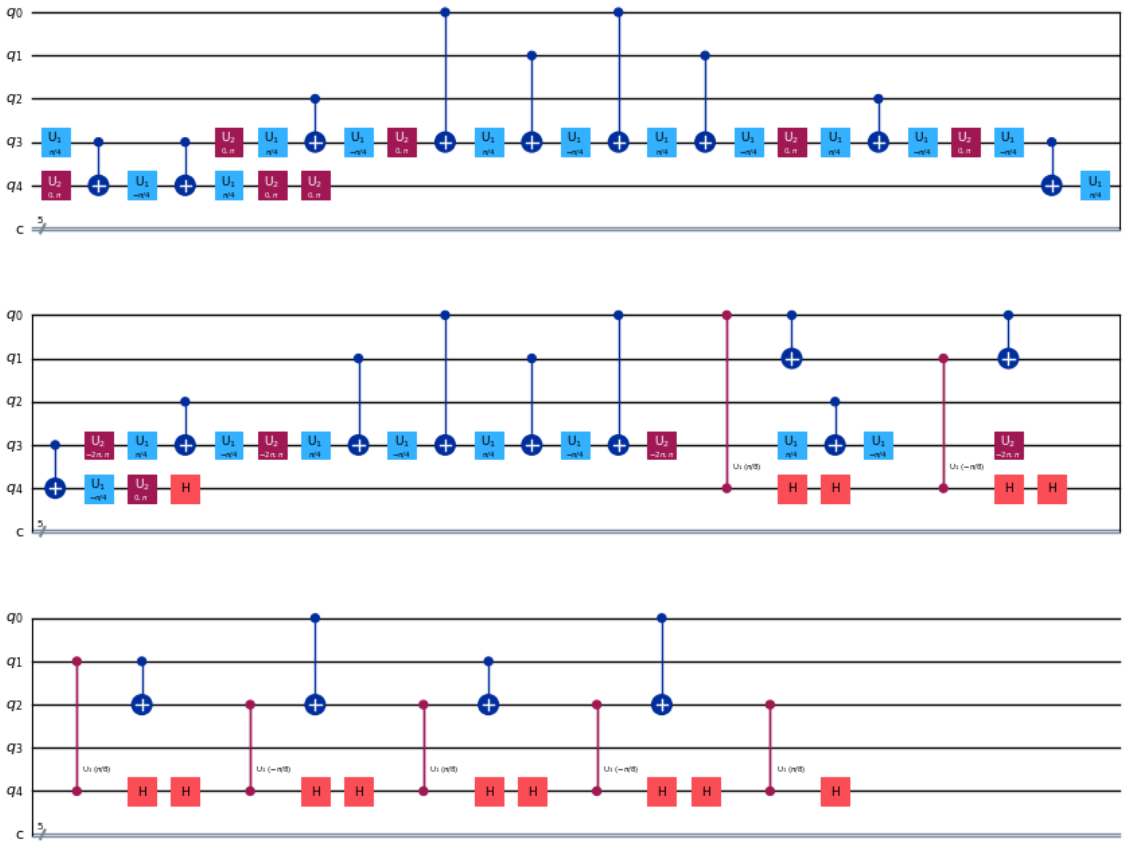


**Figure 7. Decomposition of Quantum 4-control X gate**

*Figure 6* shows the quantum 4-control X gate, which does not appear to be very different from the 3-control X gate from *Figure 4*. However, *Figure 7* shows that a far greater number of gates are required to support this functionality. Note that IBM Quantum Lab [7] and Qiskit [8] were used to generate these graphics.

# Chapter 4 Methodology

## 4.1 Qiskit and IBM Quantum Simulation

The purpose of the conducted research and experiments was to assess the possibility of using recursive circuit generation in combination with Grover's algorithm [11] to solve unstructured database search problems according to the workflow described below in *Figure 8*:

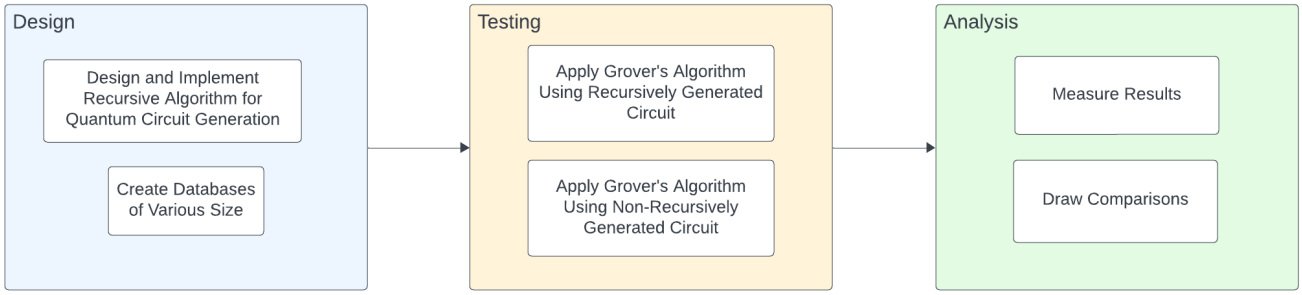


**Figure 8. Methodology Workflow**

Because of limited access to quantum hardware, the experiments were conducted in a classical computing environment with simulation capabilities. IBM Quantum Lab [7] was used to simulate all of the quantum circuits and processing. The utilized software stack comprises Python as the chosen programming language and Qiskit [8] as the selected framework for quantum computing. First, manual computation was conducted to ensure the validity and accuracy of using Qiskit and the IBM Quantum Lab. A quantum circuit representation of the classical AND gate was selected as the unit for completing this validity test because it provided the opportunity to practice stepping through the quantum gate operations by hand. The proposed circuit is shown below in *Figure 9*:

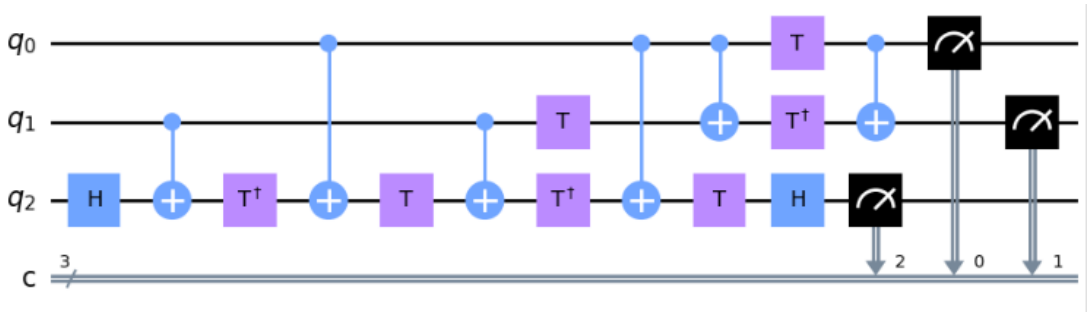


**Figure 9. Quantum Circuit Logically Equivalent to Classical AND Gate**

This test verified Qiskit's accuracy because the results matched exactly the expected values given by the truth table for a classical AND gate. The correct states were measured with 100% probability, thus ensuring the validity of using Qiskit as a framework for further work. A second proof of validity was conducted by way of the Deutsch-Jozsa algorithm [10]. Qiskit-community-tutorials was referenced for assistance with the algorithm implementation, with specific credit to online community member Paniash. Like the previous test, it performed with 100% accuracy. After Qiskit was validated as a tool which can be used with certainty, focus shifted back toward the unstructured search problem.

### 4.2 Custom Creation

One way in which circuits for the Grover's algorithm can be created is on a case-by-case basis. This involves designing an entire oracle circuit according to the necessary quantum gates to achieve that functionality and then sequencing those gates manually in code. Although this allows for increased optimization capacity because the circuit can be fit directly to the use case, in this method, there is minimal to no generalization and little opportunity for reuse. The custom creation method is standard practice for using Grover's algorithm, generally causing circuits to

expand as seen in *Figures 6 and 7* compared to *Figures 4 and 5*. This rigid method of oracle expansion may require a fewer number of total gates than a recursive oracle generation practice, but the circuits created in this fashion have limited reusability. An oracle created for a specific use case possesses a low level of flexibility because it is not be suitable for solving problem outside its original domain and scope.

### 4.3 Recursive Creation

An alternative to the custom creation approach is to utilize recursion for generating the quantum circuits necessary for implementing Grover's algorithm. In this proposed recursive method, it may be possible to gain reusability while maintaining the effectiveness of the custom creation method. It may be possible to develop a combination of quantum algorithms and classical algorithms which centers around the property of recursion to solve unstructured search problems. While this might sacrifice some level of customization, it has potential reusability. Recursively generated circuits inherently possess flexibility due to the scalable nature of recursion; a circuit can be reused over and over as the problem size increases. This principle is evidenced by the ability of 3-control X gate structure to be used in the construction of a 4-control X gate and the ability of a 4-control X gate structure to be used in the construction of a 5-control X gate, thus demonstrating reusability. The Toffoli gate was chosen as the fundamental unit for circuit generation because of its reliability. Being widely researched and utilized for quantum development, the 3-qubit Toffoli gate proves suitable for this use case. The function used for recursively generating Grover's oracle is based on the fact that the functionality of an n-control X

gate can be created by using Toffoli gates [6]. For example, a 3-control X gate can be created by using four Toffoli gates as seen below in *Figures 10 and 11*:

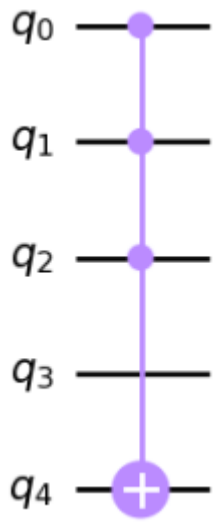


**Figure 10. 3-control X Gate**

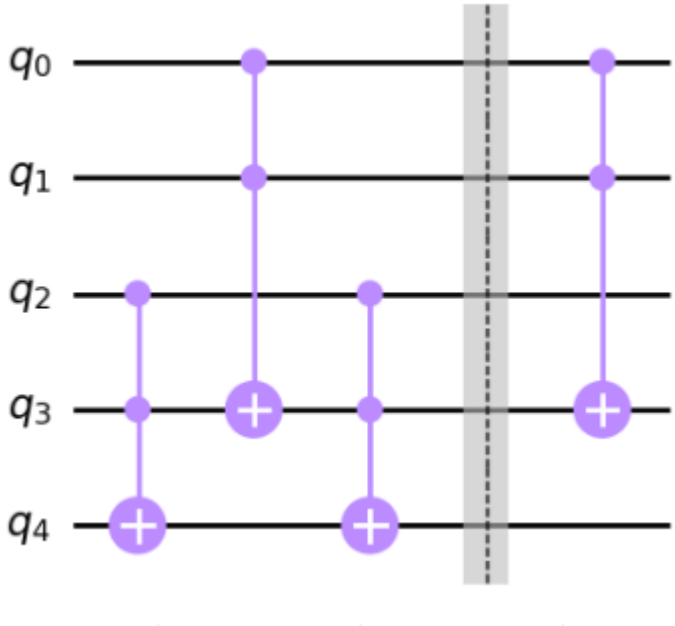


**Figure 11. 3-control X Gate Equivalent Using 4 Toffoli Gates**

A 4-control X gate can then be created by using the 3-control X gate structure with additional Toffoli gates added as described below in *Figures 12 and 13*:

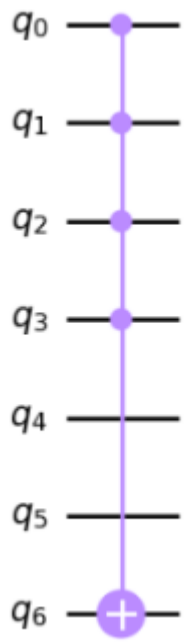


**Figure 12. 4-control X Gate**

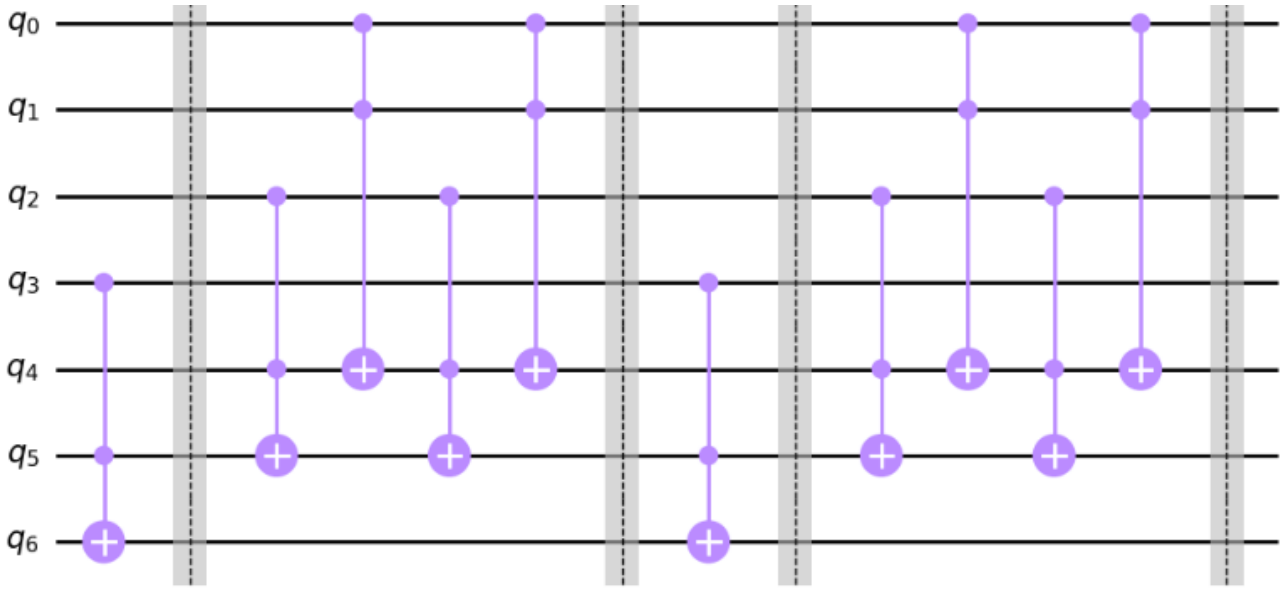


**Figure 13. 4-control X Gate Equivalent Using 3-control X Gate Structure**

### 4.4 Proposed Recursive Algorithm

Because this circuit creation is proven to be scalable [6], an n-control X gate can be constructed recursively using Toffoli gates as the base unit. *Figure 14* details the set of functions used for recursive bit-flip oracle generation using Toffoli gates.

```
# recursive function
def buildComponent(circuit,m,n,np):
    gate = QuantumCircuit(np)
    if m == 2:
        gate.ccx(0, 1,int((np+1)/2))
        return circuit.compose(gate)
    else:
        gate.ccx(m-1, n-2, n-1)
        circuit = circuit.compose(gate)
        circuit = ((buildComponent(circuit, m-1, n-1, np))).compose(gate);
        return circuit

# function to build circuit
def build(circuit,m,n,np):
    component = buildComponent(circuit,m-1,n-1,np)
    for x in range (2):
        circuit.ccx(m-1,n-2,n-1)
        circuit = circuit.compose(component)
    return circuit;

# function to build boolean oracle
def boolean_oracle(marked_state):
    n = len(marked_state[0])
    qc = QuantumCircuit(n,name='bit-flip oracle')
    qc = build(qc,int((n+1)/2),n,n)
    return qc
```

**Figure 14. Python Functions that Recursively Generate Bit-Flip Oracle**

The Qiskit code described in *Figure 14* consists of three functions that are responsible for recursively constructing a bit-flip (Boolean) oracle circuit. The *boolean_oracle* function is called

first, initializing a new quantum circuit object based on the length of the input variable: marked state. It then calls the *build* function which creates a circuit segment capable of performing the bit-flip operation as well as a separate circuit segment which ensures the oracle's reversibility by calling the recursive function *buildComponent*. The *buildComponent* function applies Toffoli gates based on specified qubit indices provided by the inputs: m and n. It calls itself recursively until the base case m = 2 is reached, meaning that the problem was reduced down to a scope which can be solved by a single Toffoli gate. After that gate is applied, the solution is perpetuated back through each recursive call, returning the main component of the oracle. This component is sandwiched between Toffoli gates to construct the bit-flip segment and then reused as the reversibility segment. Composing these segments onto one circuit produces the recursively generated bit-flip oracle.

Once a bit-flip oracle is constructed, it must be converted into a phase-flip oracle before Grover's algorithm can be applied, since the phase-flip operation is a pivotal step in the algorithm. A multi-control Z gate is appended to facilitate the conversion from bit-flip oracle to phase-flip oracle. This Z gate must be created with the marked state (solution) in mind because the element $\omega$ must receive a phase-flip when encountered by the search algorithm. The full set of Python code used to conduct experiments and gather data is given below in *Appendix A*. While this method could introduce more quantum gates to the circuit than the custom creation approach, it could prove to be more scalable and provide reusability.

# Chapter 5 Experiments and Observations

## 5.1 Expected Results

The expected results of these experiments are that Grover's algorithm will accurately locate the element ω using the recursively generated oracle in place of a non-recursively generated oracle. This accuracy is expected to span multiple database sizes. It is expected that the circuit implementing a recursively generated oracle will consist of more basic quantum gates than the circuit implementing a non-recursively generated oracle. Lastly, the computational complexity is expected to be similar between the recursively and non-recursively generated circuits.

## 5.2 Observation

Throughout these experiments, observations were recorded regarding the algorithm accuracy, number of database records, number of basic quantum gates used per circuit, and number of optimal iterations. *Figures 15 and 16* detail the accuracy of Grover's algorithm when utilizing a recursively generated oracle and a non-recursively generated oracle respectively.

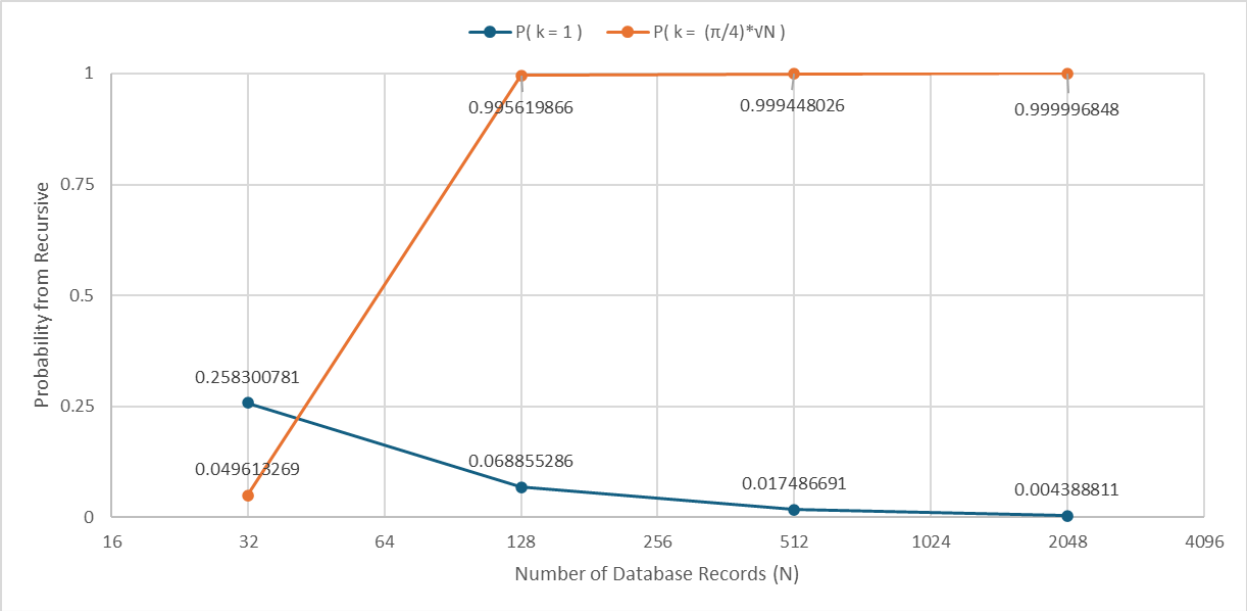


**Figure 15. Recursive Oracle Probability with k = 1 and $k \approx \frac{\pi}{4}\sqrt{N}$**

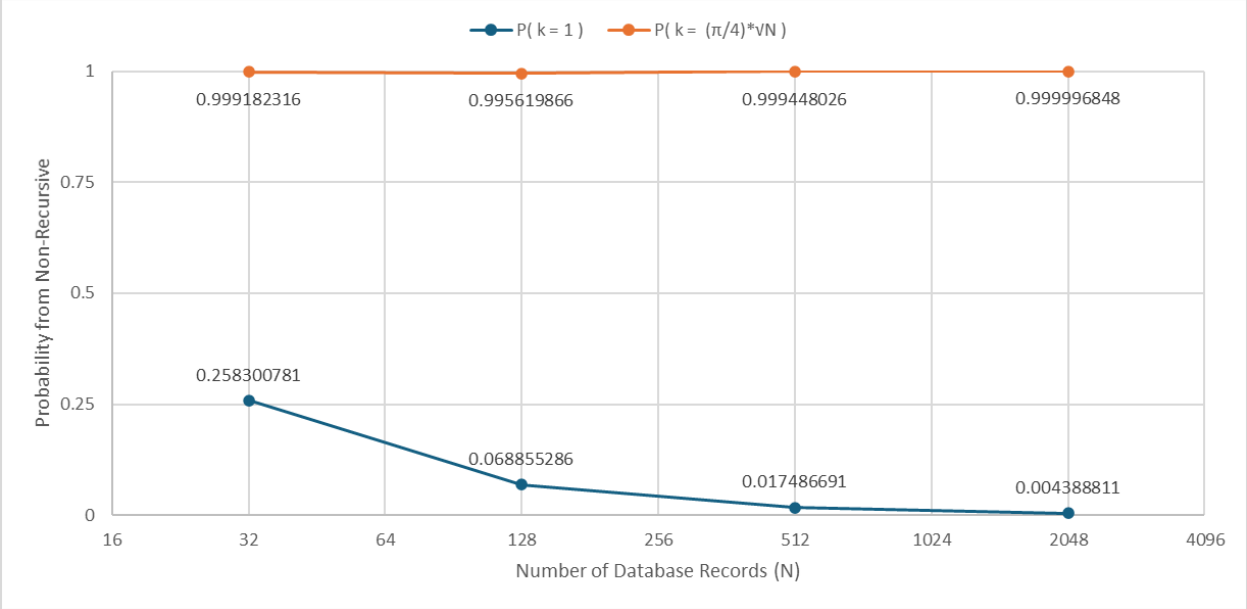


**Figure 16. Non-Recursive Oracle Probability with k = 1 and $k \approx \frac{\pi}{4}\sqrt{N}$**

*Figure 15* depicts the probability, or certainty, to which element ω was located within the database by Grover's algorithm with recursive implementation. The orange line represents the probability result after the optimal number of iterations k, as calculated by the formula $k \approx \frac{\pi}{4}\sqrt{N}$ where N is the database size in number of records. The blue line represents the probability result after only one iteration. The metric of probability corresponds to the accuracy of the circuit. One notable observation is the first orange data point—database size of 32 records—where the probability of accurately locating ω was only about 5% after 4 iterations. Although it chose the correct element ω, the probability remained extremely low. *Figure 16* describes the accuracy of the non-recursive oracle implementation, which behaved according to expectation. One notable observation is that it located element ω with 99% accuracy for every database size.

Observations were then conducted relating to the number of database records searched by the algorithm and the number of 1-qubit and 2-qubit quantum gates that comprise the recursively and non-recursively generated circuits. *Tables 1 and 2* provide data regarding those two circuit categories respectively.

**Table 1. Number of 1-qubit and 2-qubit Gates for Recursively Generated Circuit**

| Number of Database Records | Recursive 1-qubit Gates | Recursive 2-qubit Gates |
|---|---|---|
| 32 | 139 | 90 |
| 128 | 451 | 384 |
| 512 | 1415 | 1320 |
| 2048 | 4971 | 4848 |

**Table 2. Number of 1-qubit and 2-qubit Gates for Non-Recursively Generated Circuit**

| Number of Database Records | Non-Recursive 1-qubit Gates | Non-Recursive 2-qubit Gates |
|---|---|---|
| 32 | 145 | 80 |
| 128 | 379 | 336 |
| 512 | 1307 | 1248 |
| 2048 | 4827 | 4752 |

As for the recursively generated circuit data in *Table 1*, the number of database records searched ranged from 32 records to 2048 records and the number of 1-qubit gates remained greater than the number of 2-qubit gates across every database size. Both 1-qubit and 2-qubit gates increased alongside the number of database records in a linear fashion. This circuit, however, always contained two additional gates that were of larger size: $\log_2(N)$. The observations made surrounding the non-recursive circuit data collected in *Table 2* were similar. It maintained the same number of database records and a larger number of 1-qubit gates than 2-qubit gates, showcasing a similar linear progression.

### 5.3 Comparison

The comparisons made between the recursively and non-recursively generated circuits provide coverage over the list of expected results. Comparisons were made regarding algorithm accuracy, number of basic quantum gates per circuit, number of optimal iterations (computations), and number of database records. *Table 3* describes the accuracy of the two circuit types.

**Table 3. Comparison of Accuracy: Recursive and Non-Recursive**

| Number of Database Records | Recursive P($k \approx \frac{\pi}{4}\sqrt{N}$) | Non-Recursive P($k \approx \frac{\pi}{4}\sqrt{N}$) |
|---|---|---|
| 32 | 0.049613269 | 0.999182316 |
| 128 | 0.995619866 | 0.995619866 |
| 512 | 0.999448026 | 0.999448026 |
| 2048 | 0.999996848 | 0.999996848 |

The accuracy of the recursively generated circuit falls short of the accuracy provided by the non-recursively generated circuit for database size of 32 records. This data was collected using solution case ω = 00111. In this specific case, the algorithm struggled to identify the correct result, proposing x = 10111 as the next-best solution with 10% less probability than it proposed ω. However, in subsequent testing with solution case ω = 10011, the algorithm determined the result accurately with 0.999 probability. This disparity may be related to the bit-flip operation being applied to $q_5$ by the recursively generated oracle circuit, indicating a potential flaw. The algorithm's accuracy may be impacted conditionally based upon the state of the bit-flip's result qubit for the recursive algorithm implementation. Perhaps utilizing an ancillary qubit for storing the bit-flip result could solve this, but this avenue could not be explored due to time constraints. Even so, both the recursive and non-recursive circuits have identical probability for databases of size 123, 512, and 2048. They also share identical probability across every data point where the probability was measured after only one iteration (see *Figures 15 and 16*). According to these data, the recursive circuit provides an inferior accuracy compared to the non-recursive circuit for a small number of database records yet provides equal accuracy for larger databases.

*Figure 17* compares the number of 1-qubit gates present in the recursive and non-recursive circuits, and *Figure 18* likewise compares the number of 2-qubit gates between them.

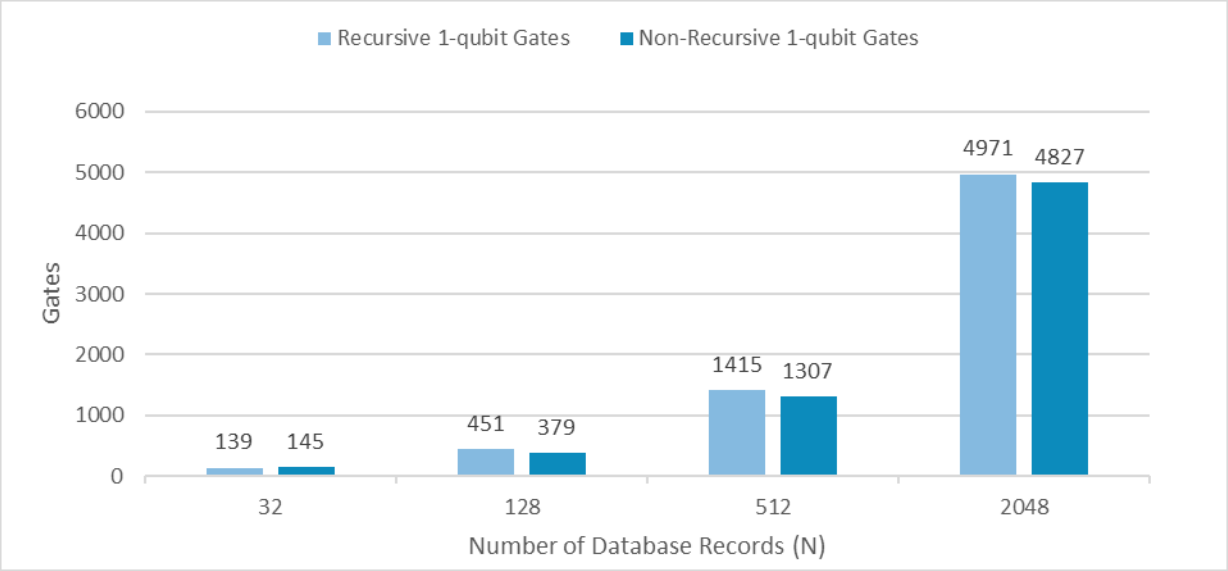


**Figure 17. Comparison of Number of 1-qubit Gates: Recursive and Non-Recursive**

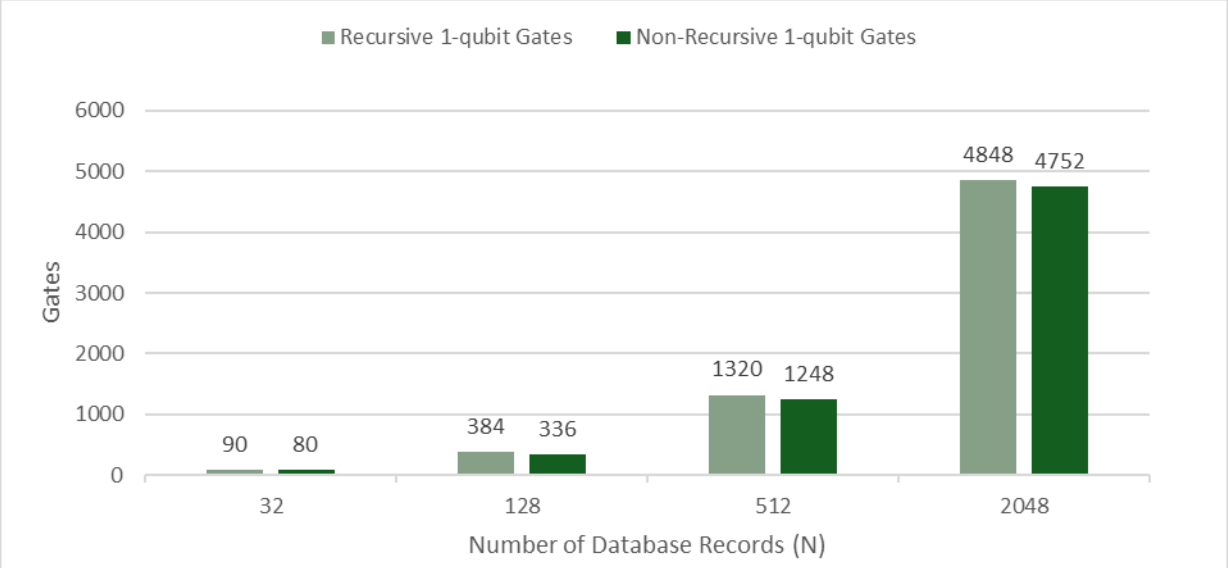


**Figure 18. Comparison of Number of 2-qubit Gates: Recursive and Non-Recursive**

At database size of 32 records, as seen in *Figure 17*, the number of 1-qubit gates for the recursive oracle is lesser than the number of 1-qubit gates for non-recursive oracle by a small margin of six gates. However, at each of the other database sizes, the opposite is true: the number of 1-qubit gates for the recursive circuit becomes greater than the number of 1-qubit gates for the non-recursive circuit. As seen in *Figure 18*, the recursively generated circuit utilizes a greater number of 2-qubit gates than the non-recursive circuit across every database size. Again, the number of both 1-qubit and 2-qubit gates increases linearly along with the number of database records.

**Table 4. Comparison of Optimal Number of Iterations: Recursive and Non-Recursive**

| Number of Database Records | Optimal Recursive Iterations | Optimal Non-Recursive Iterations |
|---|---|---|
| 32 | 4 | 4 |
| 128 | 8 | 8 |
| 512 | 17 | 17 |
| 2048 | 35 | 35 |

The comparison in *Table 4* shows that the optimal number of recursive iterations is equal between the recursively and non-recursively generated circuits. This is because both apply the same equation to calculate this optimal number of iterations, that is, $k \approx \frac{\pi}{4}\sqrt{N}$ where N is the number of database records. This represents the computational complexity of Grover's algorithm because it determines how many search iterations will be conducted; therefore, both the recursive

and non-recursive circuits have equal complexity. And because the number of database records also is identical between both circuit types, no significant comparison can be made for this metric.

### 5.3 Limitations

The present research does include certain limitations because the experiments were conducted within a simulated environment. The usage of IBM Quantum Lab [7] simulation in place of actual quantum hardware means that the study and findings may not realistically reflect the phenomena of qubits and quantum operations. Additionally, time complexity could not be included as an observation metric in this research because the simulation does not track execution time data within the quantum lab environment. Another limitation was that a simple database representation was used in place of a complex database.

# Chapter 6 Conclusions

## 6.1 Conclusion

In conclusion, this research explored the practical implementation of applying recursively generating oracle circuits to Grover's algorithm for solving the unstructured database search problem. Observations and comparisons were made according to the metrics of accuracy, number of basic quantum gates, number of optimal iterations, and number of database records. These findings suggest that recursively generated circuits are a comparable alternative to non-recursively generated circuits for implementing Grover's algorithm.

## 6.2 Challenges, Implications, and Future Work

The primary challenge during this study was the complete overhaul of the Qiskit [8] library and API that coincided with its migration to version 1.0 in early 2024. This greatly disrupted the process because a significant portion of the research and learning necessary to begin implementation using Qiskit was voided by this massive update. Certain functions and libraries became deprecated, the inheritance structure of objects was reworked, and the push occurred without fully updated documentation.

Implications to the greater research community are significant. By demonstrating the feasibility of this approach, this research may assist with efficient and scalable quantum circuit design. Future researchers could enhance the current understanding of quantum algorithms, and pioneer the development of more powerful quantum computing systems. An additional area to

investigate in future work would be the time complexity of using actual quantum hardware. This would likely shed light on how significant the difference in number of 1-qubit and 2-qubit quantum gates between recursive and non-recursive circuits is regarding time complexity of computation.

## Bibliography

[1] S. Khurana and M. J. Nene, “Implementation of Database Search with Quantum Computing: Grover's Algorithm vs Linear Search,” 2023 International Conference on Ambient Intelligence, Knowledge Informatics and Industrial Electronics (AIKIIE), Ballari, India, 2023, pp. 1-6, doi: 10.1109/AIKIIE60097.2023.10389962.

[2] E. h. Shaik and N. Rangaswamy, “Implementation of Quantum Gates based Logic Circuits using IBM Qiskit,” 2020 5th International Conference on Computing, Communication and Security (ICCCS), Patna, India, 2020, pp. 1-6, doi: 10.1109/ICCCS49678.2020.9277010.

[3] E. Osaba, E. Villar-Rodriguez and I. Oregi, “A Systematic Literature Review of Quantum Computing for Routing Problems,” in IEEE Access, vol. 10, pp. 55805-55817, 2022, doi: 10.1109/ACCESS.2022.3177790.

[4] David McMahon, “Quantum Gates and Circuits,” in Quantum Computing Explained, IEEE, 2008, pp.173-196, doi: 10.1002/9780470181386.ch8.

[5] S. P. Wang and E. Sakk, “Quantum Algorithms: Overviews, Foundations, and Speedups,” 2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP), Zhuhai, China, 2021, pp. 17-21, doi: 10.1109/CSP51677.2021.9357505.

[6] Barenco, Adriano, et al. “Elementary Gates for quantum computation,” Physical Review A, vol. 52, no. 5, 1 Nov. 1995, pp. 3457–3467, https://doi.org/10.1103/physreva.52.3457.

[7] *Quantum Experience*. IBM, 2016.

[8] *Qiskit*. IBM, 2017.

[9] Toffoli, Tommaso. “Reversible computing,” Automata, Languages and Programming. ICALP 1980. Lecture Notes in Computer Science, vol 85. https://doi.org/10.1007/3-540-10003-2_104

[10] David Deutsch and Richard Jozsa. “Rapid solutions of problems by quantum computation.” Proceedings of the Royal Society of London A. 439 (1907): pp. 553–558. doi:10.1098/rspa.1992.0167.

[11] Grover, Lov K. “A fast quantum mechanical algorithm for database search,” Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery. pp. 212–219. doi:10.1145/237814.237866

[12] Shor, Peter W. “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer,” SIAM Review, 1999, 41:2, pp. 303-332.

## Appendix A

Qiskit Code for AND gate Implementation:

```
from ibm_quantum_widgets import CircuitComposer
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(3, 'q')
creg_c = ClassicalRegister(3, 'c')
qc = QuantumCircuit(qreg_q, creg_c)

qc.h(qreg_q[2])
qc.cx(qreg_q[1], qreg_q[2])
qc.tdg(qreg_q[2])
qc.cx(qreg_q[0], qreg_q[2])
qc.t(qreg_q[2])
qc.cx(qreg_q[1], qreg_q[2])
qc.tdg(qreg_q[2])
qc.t(qreg_q[1])
qc.cx(qreg_q[0], qreg_q[2])
qc.cx(qreg_q[0], qreg_q[1])
qc.t(qreg_q[2])
qc.tdg(qreg_q[1])
qc.t(qreg_q[0])
qc.cx(qreg_q[0], qreg_q[1])
qc.h(qreg_q[2])

qc.measure(0,0)
qc.measure(1,1)
qc.measure(2,2)

qc.draw()
```

```
from qiskit_ibm_runtime import QiskitRuntimeService, Sampler

service = QiskitRuntimeService()
backend = service.get_backend("ibmq_qasm_simulator")
sampler = Sampler(backend)
job = sampler.run(qc)

job.result()
```

Qiskit Code for Deutsch-Jozsa Implementation:

```
# import libraries
import numpy as np
from qiskit import *

# create constant oracle
n = 3
constantOracle = QuantumCircuit(n+1)
output = np.random.randint(2)
if output == 1:
        constantOracle.x(n)
constantOracle.draw()

# create balanced oracle
balancedOracle = QuantumCircuit(n+1)
b_str = "101"

# create x gates
for qubit in range(len(b_str)):
    if bstr[qubit] == '1':
        balancedOracle.x(qubit)
balancedOracle.barrier()

for qubit in range(n):
    balancedOracle.cx(qubit,n)
balancedOracle.barrier()

for qubit in range(len(b_str)):
    if b_str[qubit] == '1':
        balancedOracle.x(qubit)
balancedOracle.draw()

dj_circuit = QuantumCircuit(n+1,n)

# create h gates
for qubit in range(n):
    dj_circuit = QuantumCircuit(n+1,n)
dj_circuit.x(n)
dj_circuit.h(n)
dj_circuit.draw()
```

```
# add oracle
dj_circuit += balancedOracle
dj_circuit.draw()

for qubit in range(n):
    dj_circuit.h(qubit)
dj_circuit.barrier()

for i in range(n):
    dj_circuit.measure(i,i)
dj_circuit.draw()

backend = BasicAer.get_backend('qasm_simulator')
shots = 1
results = execute(dj_circuit, backend=backend, shots=shots).result()
answer = results.get_counts()

plot_histogram(answer)
```

Qiskit Code for Recursive Grover Implementation:

```
# imports
import qiskit
import math
import matplotlib.pyplot as plt
from qiskit import *
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, transpile
from qiskit.visualization import plot_histogram
from qiskit.exceptions import MissingOptionalLibraryError
from qiskit_algorithms import Grover, AmplificationProblem
from qiskit.primitives import Sampler
from qiskit.circuit.library import PhaseOracle, GroverOperator, MCMT
```

```
# recursive function
def buildComponent(circuit,m,n,np):
    gate = QuantumCircuit(np)
    if m == 2:
        gate.ccx(0, 1,int((np+1)/2))
        return circuit.compose(gate)
    else:
        gate.ccx(m-1, n-2, n-1)
        circuit = circuit.compose(gate)
        circuit = ((buildComponent(circuit, m-1, n-1, np))).compose(gate);
        return circuit

# function to build circuit
def build(circuit,m,n,np):
    component = buildComponent(circuit,m-1,n-1,np)
    for x in range (2):
        circuit.ccx(m-1,n-2,n-1)
        circuit = circuit.compose(component)
    return circuit;

# function to build boolean oracle
def boolean_oracle(marked_state):
    n = len(marked_state[0])
    qc = QuantumCircuit(n,name='bit-flip oracle')
    qc = build(qc,int((n+1)/2),n,n)
    return qc
```

```
# define solution to search for
marked_state = ["00111"]

# number of qubits
n = len(marked_state[0])
w = int((n+1)/2)

# define Z Gate according to marked state
zGate = QuantumCircuit(n)
zGate.x(3)
zGate.x(4)
zGate = zGate.compose(MCMT('z', num_ctrl_qubits=n-1, num_target_qubits=1))
zGate.x(3)
zGate.x(4)

# create bit-flip boolean oracle based on solution
bitFlipOracle = QuantumCircuit(n)
bitFlipOracle = bitFlipOracle.compose(boolean_oracle(marked_state))

# Use Z gate to convert from bit-flip oracle into phase-flip oracle
phaseFlipOracle = QuantumCircuit(n,name='phase-flip oracle')
phaseFlipOracle = phaseFlipOracle.compose(bitFlipOracle)
phaseFlipOracle.barrier()
phaseFlipOracle = phaseFlipOracle.compose(zGate)
phaseFlipOracle.barrier()
phaseFlipOracle.draw()
#print("Phase Flip Oracle Circuit\n",phaseFlipOracle)

# calculate power
power = math.floor(math.pi/4 * math.sqrt(2**w))

# run grover's algorithm
problem = AmplificationProblem(phaseFlipOracle, is_good_state=marked_state)
#print("Grover Op",problem.grover_operator.decompose())

grover = Grover(sampler=Sampler())
result = grover.amplify(problem)
display(plot_histogram(result.circuit_results,figsize=(20,5)))
```

Qiskit Code for Non-Recursive Grover Implementation:

```
# imports
import qiskit
import math
import matplotlib.pyplot as plt
from qiskit import *
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, transpile
from qiskit.visualization import plot_histogram
from qiskit.exceptions import MissingOptionalLibraryError
from qiskit_algorithms import Grover, AmplificationProblem
from qiskit.primitives import Sampler
from qiskit.circuit.library import PhaseOracle, GroverOperator, MCMT
from collections import defaultdict
```

```
expression = "A & B & C & ~D & ~E & F & G & H & I & J & K"
power = math.floor(math.pi/4 * math.sqrt(2**11))
print("power: ",power)

oracle = PhaseOracle(expression)
problem = AmplificationProblem(oracle, is_good_state=oracle.evaluate_bitstring)
grover = Grover(iterations=power, sampler=Sampler())
#grover = Grover(sampler=Sampler())
result = grover.amplify(problem)
print(result)
#display(plot_histogram(result.circuit_results[0]))

counts = defaultdict(int)

for inst in problem.grover_operator.decompose().decompose().decompose().decompose().decompose().decompose().decompose().decompose().data:
    counts[len(inst.qubits)] += 1
print("Gates",counts)
```