5-2021

A Comparison of Quantum Algorithms for the Maximum Clique Problem

Andrew R. Haverly
arh2913@rit.edu

A Comparison of Quantum Algorithms for the Maximum Clique Problem

Andrew R. Haverly

A Comparison of Quantum Algorithms for the Maximum Clique Problem

Andrew R. Haverly

May 2021

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

R·I·T | Kate Gleason
College of ENGINEERING

Department of Computer Engineering

A Comparison of Quantum Algorithms for the Maximum Clique Problem

Andrew R. Haverly

Committee Approval:

Dr. Sonia López Alarcón *Advisor* Date
RIT, Computer Engineering Department

Dr. Amlan Ganguly Date
RIT, Computer Engineering Department

Dr. Gregory Howland Date
RIT, School of Physics and Astronomy

Acknowledgments

I would like to thank Dr. López Alarcón for her dedication to this work, Espen for being a good roommate, and my parents for their support these past four years.

Dedicated to my dog, Sammy.

Abstract

Two of the most promising computational models for quantum computing are the qubit-based model and the continuous variable model, which result in two different computational approaches, namely the qubit gate model and boson sampling. The qubit gate model is a universal form of quantum computation that relies heavily on the principles of superposition and entanglement to solve problems using qubits based on technologies ranging from magnetic fields created from superconducting materials to the spins of valence electrons in atoms. Boson sampling is a non-universal form of quantum computation that uses bosons as continuous-variable values for its computation. Both models show promising prospects for useful quantum advantages over classical computers, but these models are fundamentally different, not only on their technologies but on their applications. Each model excels in different sets of applications.

A direct comparison for solving a problem using qubit gate models and boson sampling allows one to better understand not only the individual technologies, but how to decide which model is better suited to solving a given problem and how to start development on solving the given problem. This thesis uses the maximum clique problem to examine the application development process in the qubit gate model and boson sampling as well as a comparison of other known algorithms to the maximum clique problem. The maximum clique problem is an NP-Hard problem concerned with finding the largest fully-connected subgraph. The qubit gate model algorithm to the maximum clique problem is a novel algorithm.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Quantum Computing (QC) is the method of using quantum mechanics, namely superposition, entanglement, and quantum tunneling, to perform calculations [2] [3]. QC is on track to change the world of computing in the near term [4]. It has applications in cybersecurity, medicine, and solving classically superpolynomial problems. QC has the potential to change cybersecurity by breaking standard encryption schemes we currently use and imposing new encryption schemes that cannot be broken with our current knowledge of quantum physics [5]. New medications are currently incredibly difficult to create with our slow classical algorithms to simulate molecule interactions [6]. There are several superpolynomial problems, like logistical problems, that can easily be solved using quantum algorithms, but not classical algorithms [7]. Quantum computing will change the world, but it is still a new technology and there are several models at the forefront of the race, each with their own advantages. Knowing which model is best for a particular application will allow engineers to optimize their quantum advantages.

1.2 Quantum Computing

QC solves some problems in far fewer steps than classical algorithms can ever realistically hope to achieve [5]. QC uses computing methods that are fundamentally different from classical computing. Using entanglement and superposition in the quantum gate model, a single operation can *potentially* do 2^n times more calculations than a classical computer can do with a single operation, where n is the number of qubits entangled [2]. The long term effects of quantum computing on the world are not completely clear at this point. There are frequent hardware advances and software advances that bring the reality of quantum computing being faster and cheaper than classical computing closer and closer. This thesis seeks to aid in the development of applications using different computational models of QC, namely qubit-based and continuous variable. Qubit-based quantum computing is currently at the stage of allowing flexible implementations through the use of quantum gates. Although continuous variable quantum computing also aims for flexible quantum gate implementation in the highly dimensional domain (continuous variable), the current state of the art is based on a versatile but less flexible implementation called boson sampling. Therefore, these two models, the qubit gate model and boson sampling, are the target of this work.

1.3 The Maximum Clique Problem

A clique is a fully connected subgraph. The maximum clique problem is the task of finding the largest clique in a given graph. For example, in a social network modeled as a graph, a clique would be a group of people who all know each other and a maximum clique would be the largest clique of a given population. This problem is classically hard to compute and is in the set of NP-hard problems [8]. This problem was chosen as the main subject of this research because it was relatively simple to solve

using Grover's algorithm in the qubit-based model and because there was already an algorithm using boson sampling available publicly. There are also a few other classical algorithms and an algorithm using quantum annealing described in Section 2.2.

1.4 Contribution

The maximum clique problem is an important NP-Hard problem that is still difficult to solve. This thesis proposes a QC solution to this problem with the potential of high speedup when compared to classical algorithms with the same results. This problem was chosen because of its importance in a few fields, the algorithms are relatively simple, and now there are algorithms in every form of QC examined in this thesis. The contribution of this thesis to the field of computer engineering is threefold:

1. **We created an algorithm for the maximum clique problem using Grover's algorithm using a qubit-based quantum computer.**

   It is worth noting that although Bojić proposes an outline that can be used to solve the maximum clique problem, no implementation was provided [9]. This thesis work proves that the implementation is possible.

2. **We created novel method to compare different one-hot encoded numbers with binary numbers that results in more efficient qubit utilization and circuit simulation.**

3. **We provided a unique comparison of application development using the qubit-based model of QC, boson sampling, and classical computing.**

Each of these advancements have the potential to spur further development in this field and related fields.

Chapter 2

Background

2.1 Quantum Computing

Currently, there are several models of computation using quantum properties. Two of the most promising are the qubit-based model and boson sampling. While both of these models use the properties of quantum mechanics to perform their computations, they are inherently different in the methods they use for computation.

2.1.1 The Qubit-Based Model

The qubit-based model (QB) is the most popular and widely known form of universal quantum computing. Shor's algorithm and Grover's algorithm, arguably two of the most famous quantum algorithms, were designed to use this model. This model is the method of isolating singular objects (qubits) that exhibit quantum properties. These singular objects can be the spin of the outermost electron in a phosphorus atom, the spin of a nucleus, or the magnetic field produced from superconducting materials, to name a few examples [10] [11] [12]. The qubits are then controlled through the physical implementation of a Hamiltonian that governs the time evolution of the system. The Hamiltonian is mathematically represented as a unitary operator, which can in turn be decomposed into smaller unitary operators. These are abstracted as quantum gates. These gates are very different from their classical computing counterparts.

Quantum gates interact with the qubits to change their value, entangle them, or collapse their superposition to measure their values, to name a few uses [13].

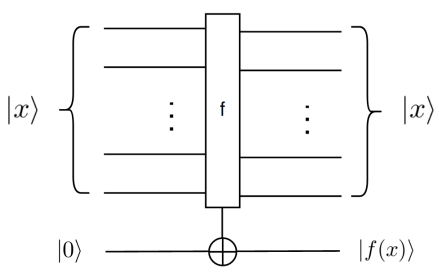
2.1.1.1 Grover's Algorithm

In this thesis, Grover's algorithm is used to solve the maximum clique problem. Grover's algorithm utilizes several properties of the qubit-based model, specifically superposition, entanglement, and phase kickback. By utilizing these properties, Grover's algorithm can be used to solve some problems much faster than classical computing can ever realistically solve these problems.

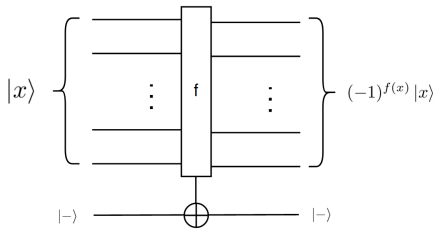
Grover's algorithm [14] was proposed by Lov K. Grover in 1996 and is one of the first quantum algorithms to show true potential in its speedups. It is a way of finding specific elements in an unsorted list of elements using a qubit-based quantum computer. Grover's algorithm can search a list of N elements using $O(\sqrt{N})$ steps, compared to the $O(\frac{N}{2})$ steps needed for the average number of comparisons to search an unordered list using a classical computer.

Grover's algorithm works by iteratively increasing the amplitude of the solution states in a superposition of states. The maximum amplitude of the probability for the solution states is achieved at $\sqrt{N}$ iterations. Grover's algorithm uses an oracle function to determine which states will be amplified in each iteration. Given a quantum register $|x\rangle$, the oracle uses quantum gates to flip the target qubit if $|x\rangle$ is a solution state. If $|x\rangle$ is not a solution state, the target qubit is unaffected. If the target qubit is in the state $|0\rangle$, then the quantum register $|x\rangle$ is left unaffected and the target qubit reacts to the function $|f(x)\rangle$, as shown in Figure 2.1a. If the target qubit is in the state $|-\rangle$, then the target qubit is unaffected and the phase kickback effect transforms the quantum register $|x\rangle$ to be $(-1)^{f(x)}|x\rangle$, depicted in Figure 2.1b.

After the oracle is run, the inverse of the oracle must be run in order to reset the oracle's ancilla qubits back to their starting states. After this, Grover's algorithm



(a) Calculating f(x) on Target Qubit



(b) Using Phase Kickback to Change the Input State of $|x\rangle$

Figure 2.1: The Effects of Phase Kickback by Changing the Target Qubit

requires a diffuser to transform the probabilities from $(-1)^{f(x)}|x\rangle$ to positive and amplified probabilities. Figure 2.2 depicts the general process of using Grover's algorithm. It shows that the oracle and diffuser combination must be repeated $O(\sqrt{N})$ times to have the largest probability of sampling a solution state.

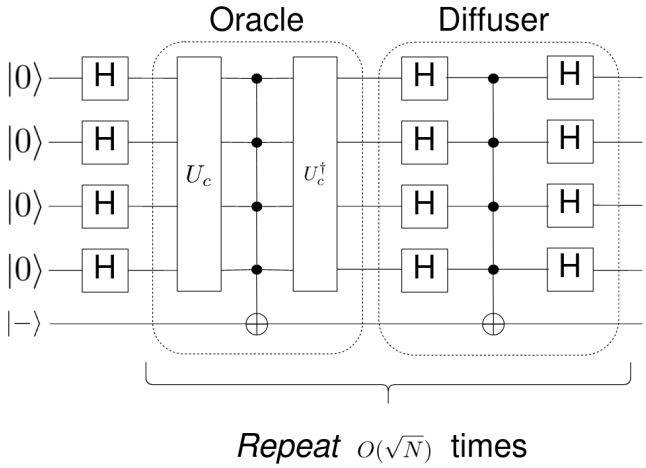


Figure 2.2: Summarizing representation of Grover's algorithm circuit. The qubit register is placed in a superposition state. The oracle applied to the superposition state results in $|f(x)\rangle$. If the target qubit is in the $|-\rangle$ state, the phase kickback effect then leaves the target qubit unaffected and changes the quantum register's states' probabilities to $(-1)^{f(x)}|x\rangle$. The diffuser —or amplitude amplifier— through multiple executions amplifies the amplitudes of the states out of the superposition that meet the oracle.

2.1.2 Boson Sampling

Boson sampling (BS) is a non-universal model of quantum computation with the ability to solve many difficult problems that quantum computers will be used to target

in the future. BS is a problem that is difficult for classical computers to simulate, but natural for bosons to solve. BS is a method of generating bosons, interfering with them to create a pattern, then sampling from the output of this interference to measure the effect of the interference on the input [15]. This can be used to perform computation. The physical implementation by the leading company Xanadu Quantum Technologies Inc. implements BS using photons [8]. They use photonic quantum hardware, linear evolution, and photodetection to perform controlled calculations. Currently, the hardware is limited to Gaussian Boson Sampling (GBS). GBS is a type of BS in which all of the gates can only perform Gaussian operations, such as displacement and squeezing. These operations keep the probability of measuring a photon in a Gaussian shape. Because of this limitation of keeping the probability of measuring a photon in a Gaussian shape, GBS cannot solve all problems and is therefore non-universal. BS is not restricted to Gaussian operations and is a universal form of quantum computing that is not limited to any set of problems. BS has a few practical uses, including the potential to speed up a major area of interest for computer scientists: graph-based algorithms. Graph-based algorithms are inherently difficult for classical computers to solve because of how many vertices and edges graphs can have. Researchers have figured out how to use boson sampling to find dense subgraphs, among other problems [8].

2.1.2.1 Boson Sampling for Dense Subgraphs

The densest k-subgraph (DkS) problem is the task of finding the densest subgraph of $k < n$ vertices with the largest density in the graph G. This problem is NP-Hard [16].

Gaussian boson sampling (GBS) has been shown to enhance classical heuristic algorithms for finding the densest k-subgraphs [17]. The core aspect of this work was to realize that GBS devices can be programmed to naturally favor sampling

dense subgraphs. GBS devices can be controlled to sample an output pattern, $S = (s_1, s_2, ...s_n)$ where s_i is the number of phtotons detected by output mode i and P(S) is the probability of S being sampled, according to

$$P(S) = |\sigma_Q|^{-\frac{1}{2}} \frac{c^2 |Haf(A_S)|^2}{p_{kcf}}$$

where A_S is the adjacency matrix corresponding to the subgraph of A associated with S, c is the user's choice of an optimization value, σ_Q is a biased covariance matrix of the Gaussian state, and $p_{kcf} := p(k \wedge cf) = p(cf|k) * p(k)$ is the collision-free probability of a photon in a mode times the probability of the number of photons per mode, where k is the number of photons per mode and cf is the Boolean value of whether or not the mode is collision free [17]. A crucial item to note is that the Hafnian of an adjacency matrix is the same as the number of perfect matchings in adjacency matrix A's graph. The number of perfect matchings in a graph is the number of unique sets of edges where every vertex is connected to exactly one edge [8].

The second item to note is that the greater the number of perfect matchings in a subgraph, the denser the subgraph is. These two facts can be used to sample dense subgraphs of a specified size.

2.2 Algorithms for the Maximum Clique Problem

A clique is a fully connected subgraph and a maximum clique is the largest clique of a graph. The maximum clique problem is the task of finding the largest clique in a given graph.

There are two main approaches to solving the maximum clique problem: brute force (exact and computationally expensive) or heuristic (approximate and computationally cheap). There are benefits and drawbacks to both approaches.

2.2.1 Brute Force Algorithm

In the worst case scenario, the brute force algorithm has to check if $2^{|V|}$ subgraphs are cliques. On average, there are $\frac{|V|}{2}$ vertices per subgraph. Depending on the density of the graph, there can be up to $\frac{|V|*(|V|-1)}{2}$ edges to check in the subgraph. Putting this all together, the classical complexity of the brute force approach requires $O(|V|^3 2^{|V|})$ operations to find the exact maximum clique. The brute force algorithm for solving this problem follows the steps:

1. Initialize $k = |V|$
2. Check if any subgraphs of size k are cliques

   If a subgraph is a clique, return the subgraph and exit.

   If there are no subgraphs of size k that are cliques, go to step 3.
3. Decrement k. Go to step 2.

Since this algorithm starts with $k = |V|$ and decrements, the maximal clique will always be found upon completion of the algorithm.

2.2.2 Simulated Annealing Algorithm

Simulated annealing is a probabilistic algorithm for approximating the global minimum for a given optimization function. This algorithm is a heuristic because the process does not guarantee finding a maximum clique. When computational speed is more important than global optimality, simulated annealing can be a good alternative to the brute force algorithm. Simulated annealing is based off of the metallurgy process of annealing in which the temperature of a metal is slowly cooled to allow the metal atoms to form a low energy crystalline structure. Simulated annealing follows this approach by having a temperature variable, T, that decreases for each iteration of the algorithm. For each iteration, a random neighboring state is selected. If the

neighboring state is more optimal, the current state is updated to the neighboring state. If the neighboring state is less optimal, there is a probability that the current state is updated to this selected neighboring state. The probability depends on the difference between the current state's optimization energy, the neighboring state's optimization energy, and the temperature T. When the temperature T is small enough, the algorithm returns its current state as the optimal solution it found [18].

The steps to finding maximum cliques using simulated annealing are:

1. Perform simulated annealing with an initial random subset of vertices σ and an optimization function shown in Equation 2.1, where G(V,E) is a graph with $|V|$ vertices, $A_G(a_{k,l}$ is the adjacency matrix of G(V,E), $k, l = 0, 1, ..., n-1$, σ is a subgraph of G(V,E) that is a clique, and m is the size of the subgraph. This objective function increases when a vertex is exchanged with a vertex of lower rank relative to the subgraph [19].

$$F(G, \sigma) = \sum_{k=0}^{m-2} \sum_{l=k+1}^{m-1} (1 - a_{\sigma(k),\sigma(l)}) \tag{2.1}$$

2. Removing the least-dense vertices until a clique is achieved. This has a worst-case time complexity of $O(|V|)$.

3. Greedily adding vertices that are fully connected to the subgraph in order to maintain the clique status for the graph [20]. This has a worst-case time complexity of $O(|V|)$.

This solution is not guaranteed to be optimal, but can certainly be much faster than the brute force algorithm. This algorithm has a worst-case time complexity of $O(|V|)$. This algorithm relates best to the boson sampling algorithm in Section 4.

2.2.3 Quantum Annealing

While not reviewed in detail in this thesis, quantum annealing is another form of quantum computing. Quantum annealing is a form of quantum computing that uses quantum physics to probabilistically find the minimum energy state of a system or energy landscape [3]. Quantum annealing uses adiabatic quantum computing. Though the gate model of quantum computing is not identical to quantum annealing, there is a polynomial time and resource mapping from the gate model approach to adiabatic quantum computing for Shor's algorithm and Grover's algorithm. The most prominent quantum annealing machines are from D-Wave. D-Wave's quantum annealers' qubits are superconducting niobium rings with magnetic fields used as qubits. To use this for computation, the energy landscape is established, and the qubits are allowed to settle. Depending on the energy landscape, the qubits usually settle into local minima or global minima. Quantum annealing is used to solve two different types of problems: optimization and sampling. Optimization problems are coverted into an energy landscape and hope to find the best or a "good enough" solution to the optimization problem. This energy landscape is often in the form of a Hamiltonian [21]. Sampling problems are those that have the goal of characterizing the energy landscape by sampling many low energy states. To use a quantum annealer to solve sampling problems, the energy landscape must be made similar to the energy landscape in question and sampled many times.

Quantum annealing has been used to solve the maximum clique problem [21]. The Hamiltonian used to solve the maximum clique problem is

$$H = H(x_1, ..., x_N) = \sum_{i \in V} a_i x_i + \sum_{(i,j) \in E} a_{ij} x_i x_j$$

where variables $x_i \in \{0, 1\}$ and coefficients $a_i, a_{ij} \in \mathbb{R}$, $V = \{1, ..., N\}$ and $E = V \times V$ [21]. The coefficients a_i and a_{ij} are the values associated to the weights of each

individual vertex and edge, respectively. This can be simplified to

$$H = -A\sum_{i=1}^{N} x_i + B \sum_{(i,j)\in\overline{E}} x_i x_j$$

where $A = 1$ and $B = 2$, when written as a quadratic unconstrained binary optimization (QUBO) problem. This equation places a negative value on the number of vertices to maximize the number of vertices favored when sampled and places a positive value on the number of edges not in the graph. Effectively, this is a cost function similar to the simulated annealing cost function, except it includes the number of vertices as well. When using this QUBO to solve the maximum clique problem, for the graph illustrated in Figure 3.1a, the solutions measured after 10 executions are AB and AC, which are the maximum cliques of the graph.

Chapter 3

Qubit-Based Model

3.1 Grover's Algorithm Outline for the Maximum Clique Problem

The approach to solve the Maximum Clique problem using Grover's Algorithm is proposed in [9] by A. Bojić. The main steps are as follows:

1. Define k as the minimum acceptable $|V|$ for the clique. Define $|x\rangle$ as the quantum register that defines the subgraph being examined. Initialize $k = 1$ and $|x\rangle = |0\rangle$.

2. Define an oracle function $f(|x\rangle)$ that flips the target qubit if the base state $|x\rangle$ represents a clique and its size is larger than the value stored in the variable k. Otherwise, $f(|x\rangle)$ does not flip the target qubit.

3. Initialize the quantum register to a superposition of every possible subgraph of the graph.

4. Run Grover's algorithm for $\sqrt{2^{|V|}}$ steps. Recall that Grover's algorithm can search through an unsorted list of N elements in $\sqrt{N}$ steps. Since there are $2^{|V|}$ elements that are being searched, one can set $N = 2^{|V|}$ for this problem. Therefore, Grover's algorithm must run for $\sqrt{2^{|V|}}$ iterations.

5. Measure the $|x\rangle$ quantum register. If the outcome is a clique with $|V| > k$, store $|x\rangle$, let $k = |V|$, and go to step 2. Otherwise go to step 6.

6. Return $|x\rangle$

In this work [9], the definition for the oracle function is mentioned but it does not provide an implementation. Section 3.2 provides this implementation.

3.2 Grover's Algorithm Oracle for the Maximum Clique Problem

The oracle for this problem needs to output a $|1\rangle$ if a subgraph is a clique AND if the subgraph is larger than k. Otherwise, the oracle needs to output a $|0\rangle$. To do this, the problem can be broken into two parts and their outputs can be combined in a logical AND using a Toffoli gate.

To start, a simple graph is chosen to explain this algorithm. A three vertex graph with two edges is best for this explanation, shown in Figure 3.1a. This graph has two maximum cliques: AB and AC. To encode this graph into the hardware, the associated edges in the q_edge register have an X gate applied to them, as shown in Figure 3.1b. This figure shows that q_edge_0, q_edge_1, q_edge_2 are associated with edges AB, AC, and BC, respectively.

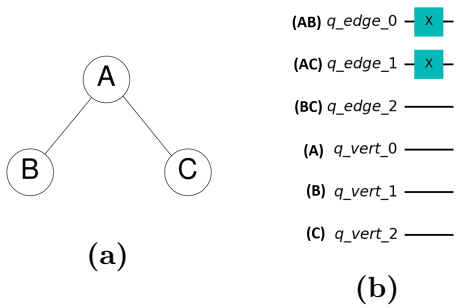


Figure 3.1: (a) Simple graph with edges AB and AC. (b) Edge and vertex encoding for the example graph.

According to Bojić's outline for this problem, a quantum register $|x\rangle$ needs to be initialized to a superposition of every possible subgraph of the graph. This can be done by applying a Hadamard gate to every qubit in the q_vert quantum register, where q_vert represents the quantum register $|x\rangle$ from Bojić's outline. As Figure 3.1b depicts, q_vert_0, q_vert_1, q_vert_2 are associated with vertices A, B, and C, respectively.

The goal of this algorithm is to identify which subsets of this graph's vertices are (a) a fully connected subgraph, or *clique*, and (b) the subgraph's $|V| \geq k$.

3.2.1 Is the subgraph a clique?

The first part of the oracle is meant to determine if the subgraph is a clique. First, for each pair of vertices, the edge is generated that would connect them *if they exist in the subgraph*. Even if that specific pair of vertices does not really exist in the graph, we want to be able to identify an edge that would connect them, which will later be used in comparison with the *actual existing edges* in q_edge to verify if the subgraph is a clique or not. A quantum register q_check_edge is created for that purpose. *If* vertices A *and* B were connected, then q_check_edge_0 should be set to $|1\rangle$, and a Toffoli gate can implement this function.

To check that all of the edges that the subgraph can possibly have existed in the graph, the values in q_check_edge to q_edge should be compared. In this case, the best way to implement this is to aim at having an output q_or=$|111\rangle$ only when the edges defined by the vertices are also existing in the q_edge register. The name q_or comes from the fact that a quantum operand behaving as a logical OR gate can achieve this behavior. If this quantum OR operand acts on the inverted q_check_edge and non-inverted q_edge, the q_or register will return $|111\rangle$ only when the subgraph in q_vert is a clique. If the outcome is anything else, then the subgraph in q_vert is not a clique. According to boolean logic, let q_edge = $|011\rangle$ and q_check_edge = $|001\rangle$,

then if we invert q_check_edge, we get $|110\rangle$. Applying a QOR gate to q_edge and q_check_edge we get $|111\rangle$. This is correct because q_check_edge corresponds to AB, and AB is a clique.

According to the logical expression, $a \vee b = \neg(\neg a \wedge \neg b)$, the implementation of the quantum or gate is depicted in Figure 3.2. We will show that this implementation does in fact produce the right results in the quantum solution of the problem.

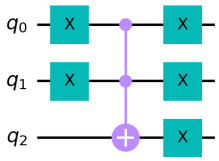


Figure 3.2: Quantum OR gate Where $q_2 = q_0$ OR q_1

The q_check_edge register is reduced to a single qubit output using two Toffoli gates as shown in Figure 3.3. The simulation results of this portion of the oracle are summarized in Figure 3.4. The qubits displayed are the *clique* qubit (1 if the subgraph is a clique and 0 if it is not) followed by the three vertices qubits *q_vert(2,0)*. For example, for the first case (0110), the vertices BC (110) do not constitute an existing subgraph or clique. On the other hand, the last case (1101) represents subgraph AC (101) which is one of the graph's cliques. The equal probabilities simply indicate that the right combination of clique+q_vert exist with approximately equal probability. This will be used later to find the largest clique.

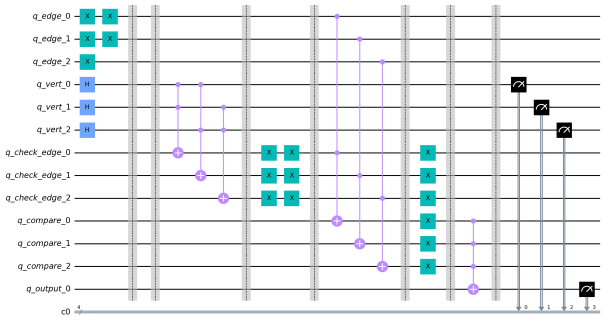


Figure 3.3: Circuit to determine if a subgraph is a clique

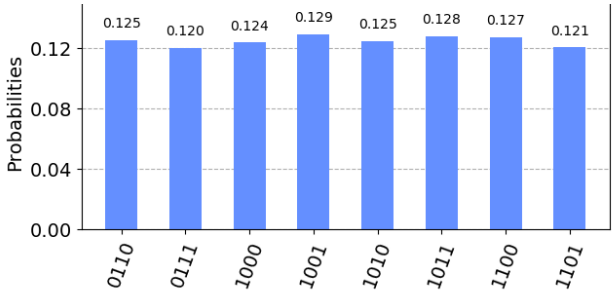


Figure 3.4: Histogram showing the simulation results of the circuit to determine if a subgraph is a clique, verifying this portion of the implementation. The X axis represents the measurement of the clique qubit (indicating whether the subgraph is a clique or not) followed by the three q_vert qubits for all possible combinations of these last three.

3.2.2 Is the subgraph larger than k?

Once the implementation is able to identify cliques in the graph, identifying the maximum clique requires comparing their sizes against an integer k, according to the algorithms described in Section 3.1. This will be done with an **integer comparator** for which the k value is updated before each run of Grovers algorithm [22].

The encoding of the vertices in a given subgraph is a one-hot encoding, meaning that each position in the string corresponds to one vertex. Since the size of the subgraph is given by the number of 1s in the encoding of the vertices, regardless of their order in the string of qubits, comparing the size requires counting the number of 1s. For example, 110, 101 and 011 are all of size two.

Comparing the number of 1s in the clique can easily be done by counting the 1s with a full adder then using an integer comparator against k. The full adder simply adds all the 1s in the one-hot encoded number of vertices into a binary number n, and the integer comparator flips a target qubit when $n \geq k$ [22]. k is updated and the comparator redefined in between every $\sqrt{2^{|V|}}$ iterations of Grover's algorithm, as explained in Section 3.1.

To implement the full adder using quantum gates, CNOT and Toffoli gates are used as shown in Figure 3.5 [23]. In this full adder, the CNOT gates in this configura-

tion are logically equivalent to a single XOR and the Toffoli gates in this configuration are logically equivalent to a single AND gate.

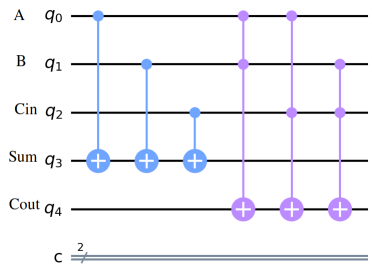


Figure 3.5: Quantum full adder

3.3 Complete Oracle

The complete oracle combines the clique identification and the size comparison with a logical AND using a Toffoli gate. This is shown in Figure 3.6.

In order to be able to use this oracle within Grover's algorithm, all of the ancilla qubits must be returned to the $|0\rangle$ state. This is done by applying the inverse of the oracle, before the final Toffoli gate. This is shown in Figure 3.7.

3.4 Complete Grover's Algorithm Implementation

In order to solve the maximum clique problem, phase kickback is applied by placing q_answer into the $|-\rangle$ to the oracle. These qubits in superposition are changed from the state $|x\rangle$ to the state $(-1)^{f(x)}|x\rangle$. As defined in Grover's algorithm, the implementation of the oracle is followed by the diffuser, which is the amplitude amplifier. This combination of steps amplifies the correct answers out the the original superposition of all possible sets of vertices. For the final implementation, the circuit has been sectioned using barriers. There are four barriers on the left and right of the oracle and diffuser. This has no impact on the circuit but helps with its visualization.

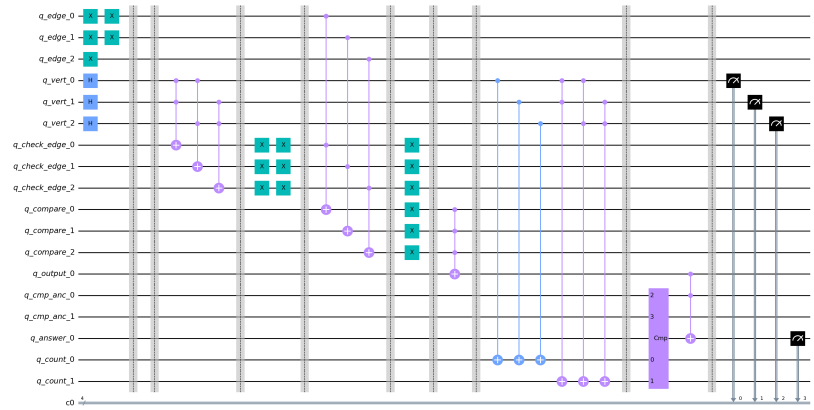


Figure 3.6: Implementation for the first half of the oracle for graphs with three vertices

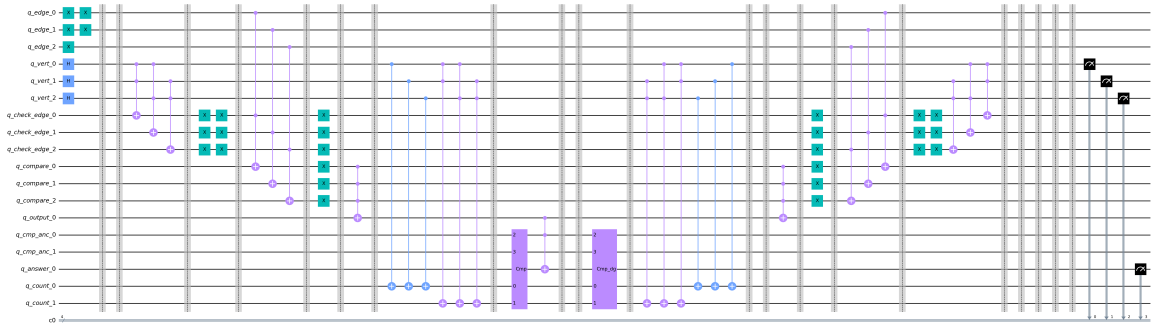


Figure 3.7: Complete oracle for graphs with three vertices

In addition, all of the elements affecting q_edge are moved before the oracle. This circuit is shown in Figure 3.8.

This circuit is regenerated and executed every $\sqrt{2^{|N|}}$ runs in order to update k with the next largest size found. The final result is shown in Figure 3.9. The AB (011) and AC (101) subgraphs are the only standing results after the execution with approximately equal probability. These are the expected results and show that this circuit correctly finds the maximum cliques in the graph.

While this implementation seems relatively straight forward, there were some challenges that needed careful thought in this work. It is important to note that the design of quantum circuits requires an understanding of the particular behavior of the quantum gates. Although certain parallelisms can be found with classical computers, such as X gates behaving

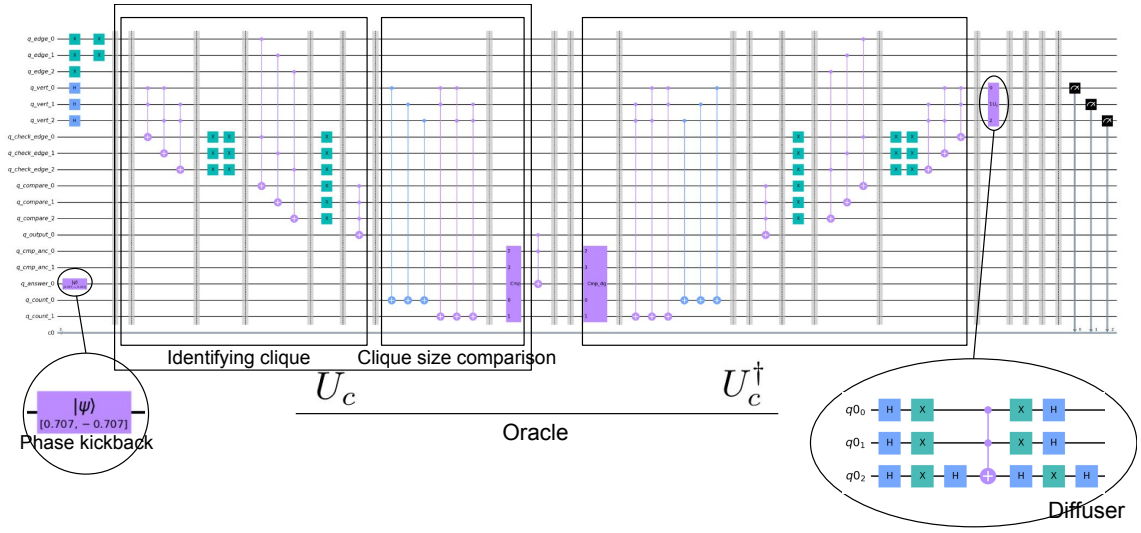


Figure 3.8: Grover's algorithm implementation to find maximum cliques. This circuit follows the structure presented in 2.2. U_c and its conjugate transpose along with the corresponding Toffoli gate in between. The qubit q_answer is initialized as $|-\rangle$. This can be done with a combination of the Hadamard and X-gate, however, since $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, this can also be expressed with a phase factor of $(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})$ as shown in the Phase kickback detail. After the oracle, the diffuser is applied, as show in the figure's detail. The redundant X-gates are shown in the circuit just for clarity, as they are part of the OR logic implementation described in Section 3.2.1

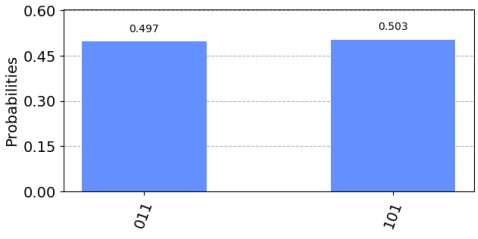


Figure 3.9: Grover's algorithm implementation to find maximum cliques simulation results. AB (011) and AC (101) are equally probable solutions for the example graph in 3.1a

as inverters or CNOT resembling XOR logic behavior, they are particular in other ways. The most notable ones being:

1. **the requirement of ancilla qubits to help with the computation's completion, while preserving information and**
2. **the understanding of the behaviors of these gates on states in superposition, in addition to the $|0\rangle$ and $|1\rangle$.**

3.5 A New Shuffling Algorithm and Conjecture

Since quantum computers currently have very few qubits, any reductions in the number of qubits required for an algorithm can be very helpful. **In this section we propose a novel, more qubit-efficient version of the oracle described above**

Currently the process of counting the number of vertices using the full adder requires additional qubits. Finding a way to perform the subgraph size comparison against k without requiring extra qubits would allow this maximum clique algorithm to be run on larger graphs while requiring fewer qubits. To do this comparison without requiring more qubits, consider the question:

Given two randomly shuffled one-hot encoded numbers, i and j, what is the probability that the binary comparison equals the one-hot comparison of i and j?

First, one needs to understand one-hot encoded numbers and binary numbers. The value of a one-hot encoded number is the number of ones in the number, e.g. the value of the number $1001_{one-hot}$ is 2 because there are two one in the number. The value of a binary number is the value of the number in base 2, e.g. the value of the number 1001_2 is $2^3 + 2^0 = 9$. To compare number using these encodings requires two different comparators: $\geq_{one-hot}$ and $\geq$ which correspond to the one-hot comparator and the binary comparator. To better understand this problem, refer to

Table 3.1: Shuffle Comparision Example with Two Bit Numbers

i	j	i $\geq_{one-hot}$ j	i $\geq$ j	Comparisons Agree
00	00	True	True	True
00	01	False	False	True
00	10	False	False	True
00	11	False	False	True
01	00	True	True	True
01	01	True	True	True
01	10	True	False	**False**
01	11	False	False	True
10	00	True	True	True
10	01	True	True	True
10	10	True	True	True
10	11	False	False	True
11	00	True	True	True
11	01	True	True	True
11	10	True	True	True
11	11	True	True	True

Table 3.1. There are 2^{2^n} combinations of i and j, when the numbers to be compared are n-bit numbers. This table shows that when i and j are two-bit numbers there are 16 possible combinations of i and j. In the third column, this table is showing the one-hot comparison between i and j (i $\geq_{one-hot}$ j). In the fourth column, this table is showing the binary comparison between i and j (i $\geq$ j). In the final column, this table is showing where these two comparisons agree. This table shows that for $\frac{15}{16}$ combinations of i and j, the one-hot comparison and the binary comparison agree.

These results are significant because they show that if two randomly shuffled one-hot encoded two bit numbers are compared using an binary comparator, there is only a $\frac{1}{16}$ chance that the binary comparator would measure incorrectly. When the number of bits in each number increases, the probability of the one-hot encoded comparison and the binary comparison being the same decreases. This decreasing function is plotted in Figure 3.10.

Shuffling Conjecture: Given two randomly shuffled one-hot encoded numbers i and j, the probability that the binary comparison equals the one-hot comparison of

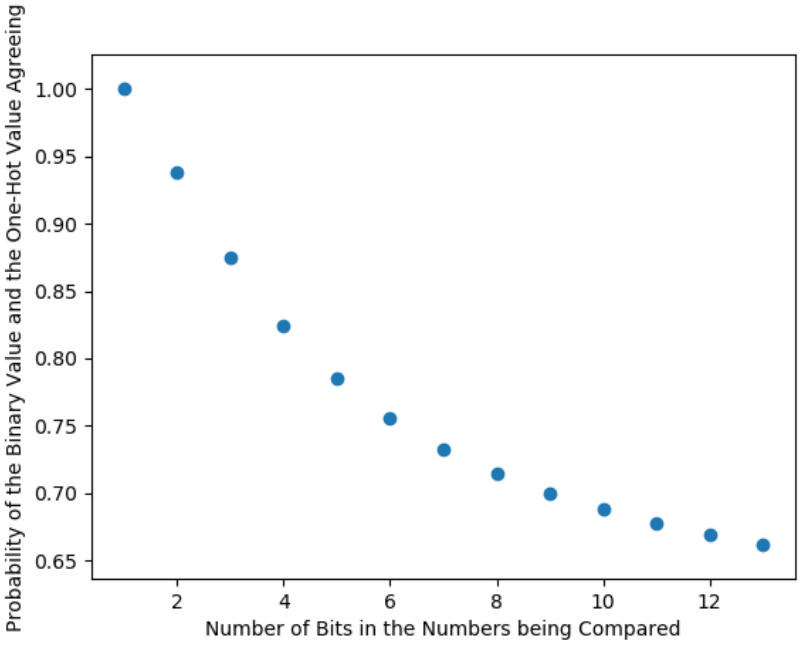


Figure 3.10: Comparison agreement probabilities as the number of bits in each number increases

i and j is greater than $\frac{1}{2}$ for all lengths of the two numbers.

The shuffling Conjecture is a novel contribution, especially important in the NISQ era [4]. The NISQ era is currently severely limited by the number of qubits available in current hardware. By providing a way to reduce the number of qubits required for a circuit to run, the Shuffling Conjecture can be used for many problems in the near future.

3.6 Applying the New Shuffling Algorithm to Grover's Algorithm for the Maximum Clique Problem

Assuming that the Shuffling Conjecture is correct, this can be integrated into the oracle implementation for the maximum clique problem, described in Section 3.2.

To compare two one-hot encoded numbers, the numbers i and j must first be identified. With this problem, i can represent the number of vertices in q_vert, which

represents the number of vertices in the subgraph. The number j can represent the k value represented as a one-hot encoded number. Looking at Figure 3.10, one can gather that the probability of the correct one-hot encoded comparison agreeing with the integer comparison is 87.5%. To further understand how this shuffling will affect the outcome of the program, one can evaluate the probabilities of the comparisons agreeing with respect to the difference of the number of ones in the one-hot encodings, as Figure 3.11 depicts. This figure shows that when the two numbers have the same number of ones there is a $\frac{2}{3}$ chance that the one-hot comparison and the integer comparison agree. When there is a single bit difference, there is a $\frac{8}{9}$ chance that the two comparisons agree. When there is a two bit difference, there is a 100% chance that the two comparisons agree.

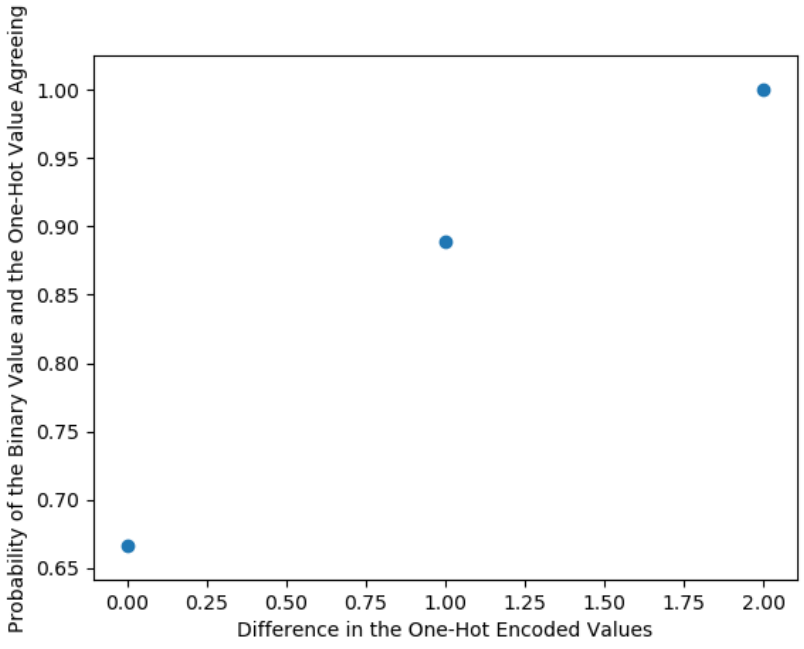


Figure 3.11: Comparison agreement probabilities as the difference of the one-hot encoded numbers increases

Since the simulations run in this paper use many trials, the shuffling can be declared classically for each run. This means that when the circuit is being defined, swap gates can be inserted before and after the integer comparator. This is depicted

in Figure 3.12.

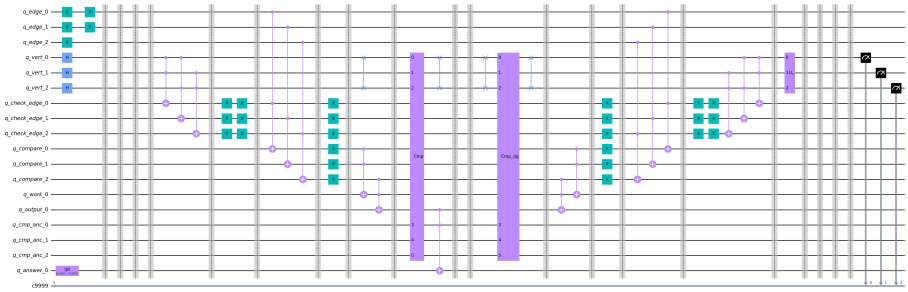


Figure 3.12: Grover's algorithm to solve the maximum clique problem using the novel shuffling algorithm

The simulation results for the circuit shown in Figure 3.12 are shown in Figure 3.13. Unlike the implementation of Grover's algorithm that used counting to compare the size of the subgraph to the value k, these results are not perfect. These results do, however, show that there is approximately a 70% chance that the correct value will be measured. The values with a single one in their one-hot encoding have a much smaller chance of being measured because $\frac{8}{9}$ times their comparison when $k = 2$ registers as smaller than k. The rest of the values are not cliques so they have a smaller probability of being measured.

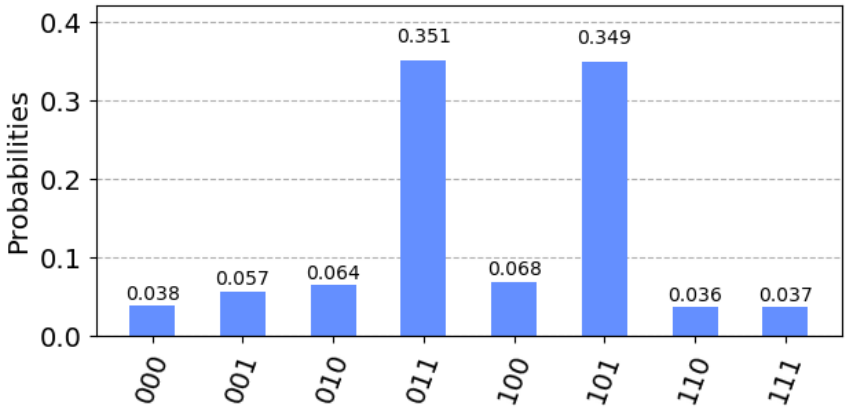


Figure 3.13: Simulation results for Grover's algorithm to solve the maximum clique problem using the novel shuffling algorithm

3.7 Finding the Maximum Clique using Grover's Algorithm for a Graph with Four Vertices using the Shuffling Algorithm

This algorithm works for all graphs, not just three vertex graphs. To run the algorithm on a four vertex graph, more qubits are needed. As an example, examine the four vertex graph in Figure 3.14. There are two maximal cliques: ABD and ACD.

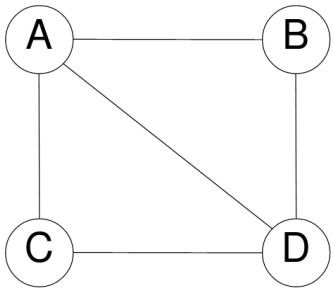


Figure 3.14: Example Four Vertex Graph

To encode this graph into qubits, q_vert_0, q_vert_1, q_vert_2, and q_vert_3 correspond to vertices A, B, C, and D, respectively. Also, q_edge_0, q_edge_1, q_edge_2, q_edge_3, q_edge_4, and q_edge_5 correspond to edges AB, AC, AD, BC, BD, and CD, respectively. The complete circuit for this algorithm is shown in Figure 3.15. Simulating this circuit produces the histogram shown in Figure 3.16.

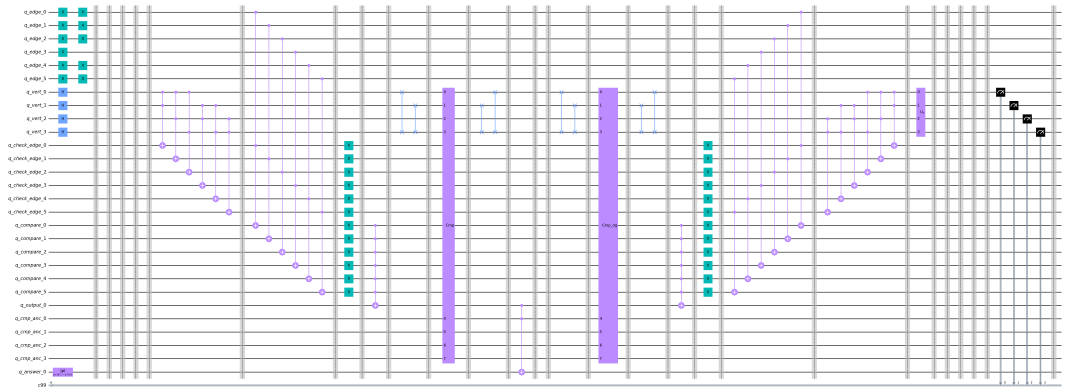


Figure 3.15: Circuit for Four Vertices

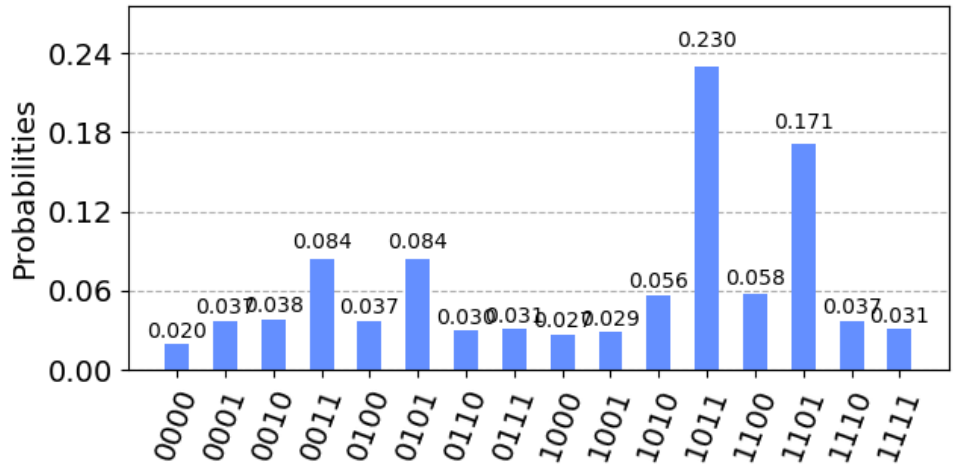


Figure 3.16: Simulation Results for the Four Vertex Circuit

These simulation results show that the circuit produces the output of ABD and ACD with the largest probabilities. These are the maximum cliques for the graph, so these results are expected. It is important to note that this solution uses far fewer qubits than the solution that uses counting to determine with 100% certainty that $|V| \geq k$.

It is important to note that this shuffling approach reduced the circuits complexity to the point of making the simulation possible, a simulation that was not possible otherwise. This simulation ran successfully on a system with 32GB of memory. This is because the shuffling version of this algorithm requires fewer qubits than the version that counts the number of vertices. The counting version of the algorithm was also implemented, shown in Figure 3.17. This circuit could not be simulated on the same system that the shuffling version of the circuit was simulated on. This is because the counting version of the circuit has more qubits and requires more memory to simulate. This circuit required more than 256GB of memory to run successfully. This was not accessible at the time of this development.

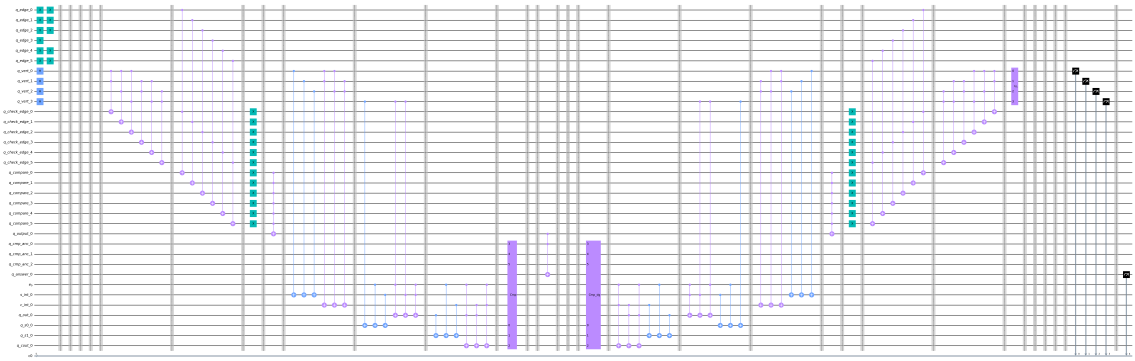


Figure 3.17: Counting Version of the Circuit for Four Vertices

3.8 Quantum Cost of the Grover's Algorithm Implementation

To understand the current and future capabilities of this algorithm, the quantum cost of the counting version of the circuit, shown in Figure 3.7, is analyzed. The shuffling version of the circuit is analyzed after this analysis. The two most important factors in the NISQ era are the number of qubits required and the depth of the circuit. The circuit is analyzed as-is. This circuit is not decomposed into primitive gates or optimized for execution. However this is still a good estimation of the scalability of this implementation.

3.8.1 Number of Qubits Required

Before determining the number of qubits required, it is important to understand the scalability of the number of edges with the size of the graph. There are $|E| = \frac{|V|*(|V|-1)}{2}$ edges in any fully connected graph. As an upper bound, this is $O(n^2)$ where n is the number of vertices in the graph. This is an important upper limit to take into account since finding the maximum clique involves exploring all possible edges in the graph for $|V|$ vertices, whether those are truly connected or not in the specific graph.

The breakdown of qubits is as follows:

- q_edge: This register scales directly with the number of edges in the graph. This register requires $|E|$ qubits.
- q_vert: This register scales directly with the number of vertices in the graph. This register requires $|V|$ qubits.
- q_check_edge: This register requires $|E|$ qubits.
- q_compare: This register requires $|E|$ qubits.
- q_output: This register requires 1 qubit.
- q_cmp_anc: This register requires $\lceil \log_2(|V|) \rceil$ qubits.
- q_answer: This register requires 1 qubit.
- q_count: This register requires $O(|V| * \log_2(|V|))$ qubits. Conceptually this can be understood by two facts:
  1. $|V|$ can be represented with $\log_2(|V|)$ qubits.
  2. Performing the addition operation for each qubit in q_vert would require $|V|$ full adders.
  3. This can be optimized by removing extra full adders in steps that cannot have that large of an output. For example, if the operation is on the third vertex being added, only two qubits are required for the output.

  For a graph with three vertices, q_count requires two qubits. For a graph with four vertices, q_count requires six qubits.

Putting this all together, the circuit requires

$$\frac{3 * |V| * (|V| - 1)}{2} + |V| + \lceil \log_2(|V|) \rceil + O(|V| * \log_2(|V|)) + 2$$

qubits to use this algorithm for a graph with $|V|$ vertices, which is $O(|V|^2)$ qubits.

For a graph with three vertices, this algorithm requires 18 qubits. For a graph with four vertices, this algorithm requires 34 qubits. Figure 3.18 summarizes the potential growth of this metric with the number of vertices.

The shuffling version of this solution very similar, for the most part. The differences are that q_cmp_anc requires $O(|V|)$ qubits and q_count is unnecessary. This results in a total qubit requirement of

$$\frac{3 * |V| * (|V| - 1)}{2} + 2 * |V| + 2$$

qubits, for a graph with $|V|$ vertices, which still has the complexity of $O(|V|^2)$, but has a smaller actual value as $|V|$ scales.

3.8.2 Depth of Circuit

The depth of the circuit is the other important metric for this algorithm. It is important in this case to keep in mind that in the real hardware implementation, the compiler will generate more efficient hardware through optimization and sufficient scheduling and mapping. For this paper, the calculations bellow have only applied obvious optimization such as the removal of redundant X-gates. However, this is nonetheless a good estimate of the scalability of this implementation, given that the final resulting implementation will only experience a proportional reduction of the number of levels in the circuit discussed here. Again, the full detail of the depth calculation is not described for the sake of space. It is important to notice that U_c and ${U_c}^\dagger$ have the same depth. Within these, some non-obvious portions of the circuit are the integer comparators, which require $5 * (|V| - 2)$ logic levels. This can be analyzed step-by-step as with the number of qubits:

1. The first two levels of operations do not grow with the size of the graph.

2. The levels to determine all of the possible edges in the given subgraph scale with $|E|$.

3. The levels with the double X gates are redundant and unnecessary. They are only included in this paper to aid understanding.

4. The next three levels scale with $|E|$.

5. The X gates after the three Toffoli gates do not grow with the size of the graph.

6. The multiple control Toffoli gate requires 1 level.

7. The levels used for counting require three times more levels than the number of qubits required for q_count.

8. The integer comparator requires $5 * (\log_2(|V|) - 2)$ levels.

9. The Toffoli gate that targets the q_answer qubit requires 1 level.

10. The next few levels are a reverse of the first part of the oracle. This means we have to add on the depth of our circuit equal to the length of the levels described in 2-8.

11. Next, we have our diffuser. This requires seven levels.

12. Finally, we measure the qubits. This scales with $|V|$.

In total, this oracle requires

$$2 * |V| * (|V| - 1) + O(|V| * \log_2(|V|)) - 7$$

levels to operate correctly and the pre- and post-processing number of levels required is $|V| + 1$. So, for one iteration of the oracle the depth of the circuit is $O(|V|^2)$ in the worst case scenario. For a graph with three vertices, 57 levels are required for a

single iteration of Grover's Algorithm. For a graph with four vertices, 98 levels are required. The number have been verified correct for the four vertex graph. Again, Figure 3.18 summarizes the potential growth of the circuit's depth with the number of vertices.

The shuffling version of this solution very similar, for the most part. The differences are that levels for counting the vertices are now used to shuffle the vertices, which requires $O(|E|)$ levels and the integer comparator requires $5 * (|V| - 2)$ levels. This results in a total level requirement of

$$3 * |V| * (|V| - 1) - 7$$

levels, for a graph with $|V|$ vertices. This still has a complexity of

q_cmp_anc requires $O(|V|)$ qubits and q_count is unnecessary. This results in a total qubit requirement of

$$\frac{3 * |V| * (|V| - 1)}{2} + 2 * |V| + 2$$

qubits, for a graph with $|V|$ vertices, which still has the complexity of $O(|V|^2)$, but has a larger actual value as $|V|$ scales.

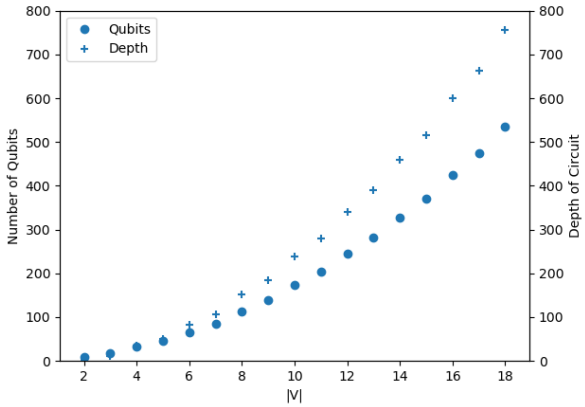


Figure 3.18: Number of qubits and circuit depth growth with the number of vertex in he graph, according to this implementation

Chapter 4

Boson Sampling

4.1 Using Boson Sampling to Identify Locally Maximal Cliques

As described in Section 2.1.2.1, GBS can be used to sample dense subgraphs from a graph. While this may produce subgraphs that are very similar to cliques, they are not guaranteed to be cliques, or even locally maximally dense subgraphs. This is when the second and third steps of the simulated annealing algorithm from Section 2.2.2 can be used. To recap, the the second step of the simulated annealing algorithm is to shrink the dense subgraph into a clique and the third step is to grow the clique until there are no more vertices possible to add [20].

Since Xanadu Quantum Technologies, Inc. provides cloud access to their devices with special permissions, sampling dense subgraphs from small graphs can be implemented and run on an actual device. Although the publicly available device has its limitations, it can still be useful. Right now, it is limited to sampling from bipartite graphs [1]. The physical device uses beamsplitters, squeezers, interferometers, and Fock basis measurement gates in a configuration shown in Figure 4.1.

There are several steps required to sample dense subgraphs using the BS device:

1. Extract the adjacency matrix, B, for the graph from which dense subgraphs will be sampled. Pad the adjacency matrix with zeros until it is a 4x4 matrix.

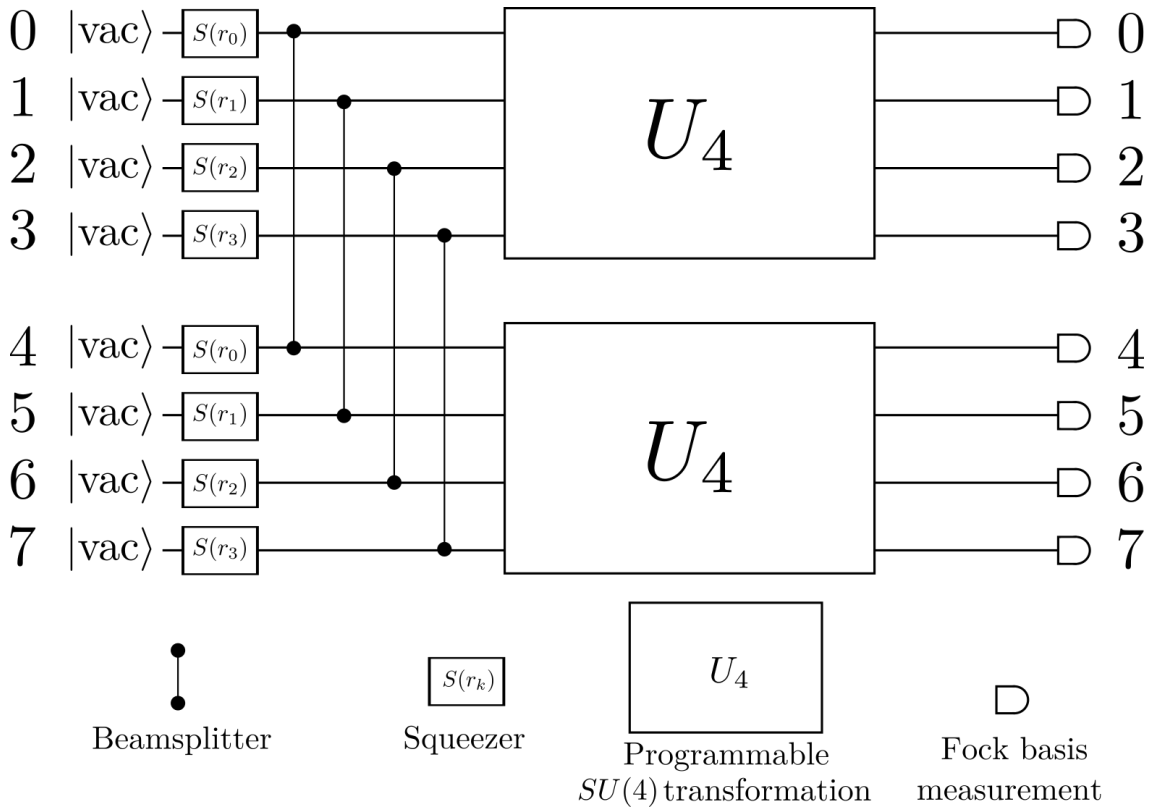


Figure 4.1: Xanadu X8 device configuration [1]

2. Convert this adjacency matrix into the form $A = \begin{bmatrix} 0 & B \\ B^T & 0 \end{bmatrix}$.

3. Determine the mean photon number per mode. The mean photon number per mode is the total number of photons divided by the total number of modes. This value indicates how many photons the BS device needs to create when given an exact number of modes. Currently, the best method of finding this value is through trial-and-error from the output of the embedding program to get either a value of 0 or 1 photon for each mode. To determine a valid mean photon number per mode, a starting point must be determined. A good starting mean photon number per mode is to start with the density of the graph. Next, test if the embedding program determines that the squeezing amplitude is invalid, either indicating that the mean number of photons per mode is too low or too

high. Iteratively change the mean number of photons per mode until the mean number of photons per mode is valid, determined by the embedding program. This can also be found by backtracking the calculations from the end values of 0 or 1 photon for each mode.

4. Embed the bipartite graph using the publicly available software [8].

5. Compile the program into the blackbird quantum photonics programming language [24].

6. Run the device for many samples and take the mean photon count per mode. There is a trade-off between the compute time and the accuracy that an end user must determine is ideal, so there is no exact best number of samples to retrieve.

7. Take the k largest values from the first half of the sampling output. The vertices associated with these largest values are the vertices associated with the densest subgraph of size k.

Following these steps for the graph shown in Figure 3.1a, the resulting B matrix is: $B = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$. This matrix was built by taking the adjacency matrix of the graph, which is: $\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$ and then zero-padding this matrix until it is a 4x4 matrix.

For this graph, the successful mean number of photons per mode is 0.690566666666664. This was determined by taking the density of the graph, $\frac{2}{3}$, and iteratively adjusting the mean number of photons per mode until a valid value was achieved. The embedded bipartite graph is compiled into the blackbird language. This program

was compiled using Xanadu Quantum Technologies, Inc.'s Blackbird compiler [24]. This listing shows rotation gates (Rgate), two-mode squeeze gates (S2gate), Mach-Zehnder interferometers (MZgate), and measurement operations (MeasureFock). The qumodes used are Gaussian probability landscapes with respect to the photons' momentum and position. The **rotation gate** rotates this landscape a specified angle. **Single mode squeeze gates** decrease the uncertainty for either the position or the momentum and therefore increase the uncertainty for either the momentum or the position, respectively, according to the Heisenberg uncertainty principle. **Two-mode squeeze gates** entangle two photons and perform a similar squeezing operation on the two qumodes according to the Heisenberg uncertainty principle. **Interferometers** use the interference, either constructive or destructive, of two modes to produce two separate probability landscapes from the two input probability landscapes. **Fock measurement** operations use photon counters to count the number of photons received at the output of each qumode. These instructions follow the format of the architecture shown in Figure 4.1.

Listing 4.1: Compiled Blackbird Code to Sample from a Bipartite Graph Embedding

```
S2gate(1, 0) | (q[0], q[4])
S2gate(1, 0) | (q[1], q[5])
S2gate(0, 0) | (q[2], q[6])
S2gate(0, 0) | (q[3], q[7])
MZgate(3.142, 0) | (q[0], q[1])
MZgate(0, 0) | (q[2], q[3])
MZgate(3.142, 1.571) | (q[1], q[2])
MZgate(1.571, 0) | (q[0], q[1])
MZgate(3.142, 2.356) | (q[2], q[3])
MZgate(1.571, 0) | (q[1], q[2])
Rgate(3.927) | (q[0])
Rgate(1.571) | (q[1])
Rgate(1.571) | (q[2])
```

```
Rgate(0) | (q[3])
MZgate(3.142, 0) | (q[4], q[5])
MZgate(0, 0) | (q[6], q[7])
MZgate(3.142, 1.571) | (q[5], q[6])
MZgate(1.571, 0) | (q[4], q[5])
MZgate(3.142, 2.356) | (q[6], q[7])
MZgate(1.571, 0) | (q[5], q[6])
Rgate(3.927) | (q[4])
Rgate(1.571) | (q[5])
Rgate(1.571) | (q[6])
Rgate(0) | (q[7])
MeasureFock | (q[0], q[1], q[2], q[3], q[4], q[5], q[6], q[7])
```

Running this on the X8 device produces the results in Listing 4.2. Each row of the matrix represents a sample from the X8 device. Each value in the row represents the number of photons measured per mode for that sample. Taking the mean number of photons per mode for these samples produces the results: [1.2 0.4 0.4 0. 0.5 0.4 0.4 0.]. Taking the 2 largest values from the first half of the array gives the result of AB or AC, which are the densest subgraphs with $|V| = 2$ from the three node graph in Figure 3.1a. Although this is trivial, shrinking (step 2) and growing (step 3) the graph according to the steps outlined in Section 2.2.2 produce the locally maximal cliques of AB or AC, which are the maximum cliques in the graph.

Listing 4.2: Samples from the X8 Device

```
[[1 0 0 0 0 0 0 0]
 [2 0 1 0 1 0 1 0]
 [0 1 1 0 0 0 0 0]
 [0 1 1 0 1 0 1 0]
 [3 1 0 0 0 1 1 0]
 [0 0 1 0 2 0 1 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
```

```
[6 0 0 0 0 3 0 0]
[0 1 0 0 1 0 0 0]]
```

4.2 Finding Maximum Cliques by Simulating a More General Version of Gaussian Boson Sampling

Xanadu Quantum Technologies, Inc. provides a framework called Strawberry Fields (SF) that can be used to simulate more general architectures of GBS. This framework is much more flexible and can be used to generate dense subgraphs from a graph. The steps to find the maximum cliques using this simulator are similar to the steps outlined in Section 4.1. As a working example, assume the graph that the dense subgraphs are being sampled from is the three node graph shown in Figure 3.1a. The steps are as follows:

1. Extract the adjacency matrix, A, for the graph.
2. Declare the mean number of photons per mode as the requested size of the dense subgraph.
3. Embed the graph using the publicly available software [8].
4. Compile the program into the blackbird quantum photonics programming language [24].
5. Run the device for a few samples and take the mean photon count per mode.
6. Take the k largest values from the sampling output. The vertices associated with these largest values are the vertices associated with the densest subgraph of size k.

Following these steps, the adjacency matrix, A, is $A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$

The compiled program is shown in Listing 4.3. This compiled program uses squeeze gates (Sgate), rotation gates (Rgate), beamsplitters (BSgate), and measurement gates (MeasureFock).

Listing 4.3: Compiled Blackbird Code to Simulate Sampling from a Graph Embedding

```
Sgate(-0.8814, 0) | (q[0])
Sgate(-0.8814, 0) | (q[1])
Rgate(1.571) | (q[0])
BSgate(0.7854, 0) | (q[0], q[1])
Rgate(5.084) | (q[0])
Rgate(4.712) | (q[1])
Rgate(1.571) | (q[2])
BSgate(-0.7854, 0) | (q[1], q[2])
Rgate(-3.142) | (q[1])
Rgate(2.77) | (q[0])
MeasureFock | (q[0], q[1], q[2])
```

After sampling from the simulator 1000 times, the mean number of photons measured per mode are [0.483 0.316 0.323]. Since the simulated values of 0.316 and 0.323 are so similar, it can be assumed that they are approximately equally likely to be sampled. Taking the two largest values values yields the subgraphs AB and AC. Performing the shrinking (step 2) and growing (step 3) operations from Section 2.2.2. produces the locally maximal cliques of AB and AC, which are the maximum cliques of the graph.

To use BS to find the maximum clique of the graph shown in Figure 3.14, the

adjacency matrix must be identified as $A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$ and the adjacency matrix must be embedded in the same process as before. After sampling from the simulator 1000 times, the mean number of photons measured per mode are [0.501 0.321 0.333 0.491]. Since the values 0.501 and 0.491 are very close, they can be assumed to have the same sampling probability. Since the values 0.321 and 0.333 are very close, they can be assumed to have the same sampling probability. Taking the three largest values yields the subgraphs ABD and ACD. Performing the shrinking and growing operations from 2.2.2 produces the locally and globally maximal cliques of ABC and ACD.

4.3 Quantum Cost

4.3.1 Xanadu X8 Device Program Quantum Cost

Currently, all programs run on the X8 device have the same quantum cost [1]. There are 8 modes used for each program. Additionally, the layers for this program are the first squeezing layer, four beam splitters, a rotation layer, four more beam splitters, another rotation layer, and finally the measurement layer. This results in a total of 12 layers for all programs run on the X8 device. These values do not scale because this is a fixed-size architecture and can only run on graphs with a maximum of four vertices.

4.3.2 Simulated Program Quantum Cost

The simulated program requires 3 modes. This is exactly the number of vertices in the graph. Therefore, the number of modes required to sample dense subgraphs from

a graph is $|V|$.

Additionally, the layers for this program are the first squeezing layer, a rotation layer, a beam splitter, another rotation layer, another beam splitter, a final rotation layer, and a measurement layer. Therefore, for the graph in Figure 3.1a this program requires 7 layers to sample its dense subgraphs.

Varying the density of the graph and the size of the graph produces different depths required. Figure 4.2 shows the depth of the quantum circuit for different densities and sizes of graphs. This figure shows that the density of the graph does not have a large effect on the depth of the quantum program but the size of the graph does have a large effect on the depth of the quantum program.

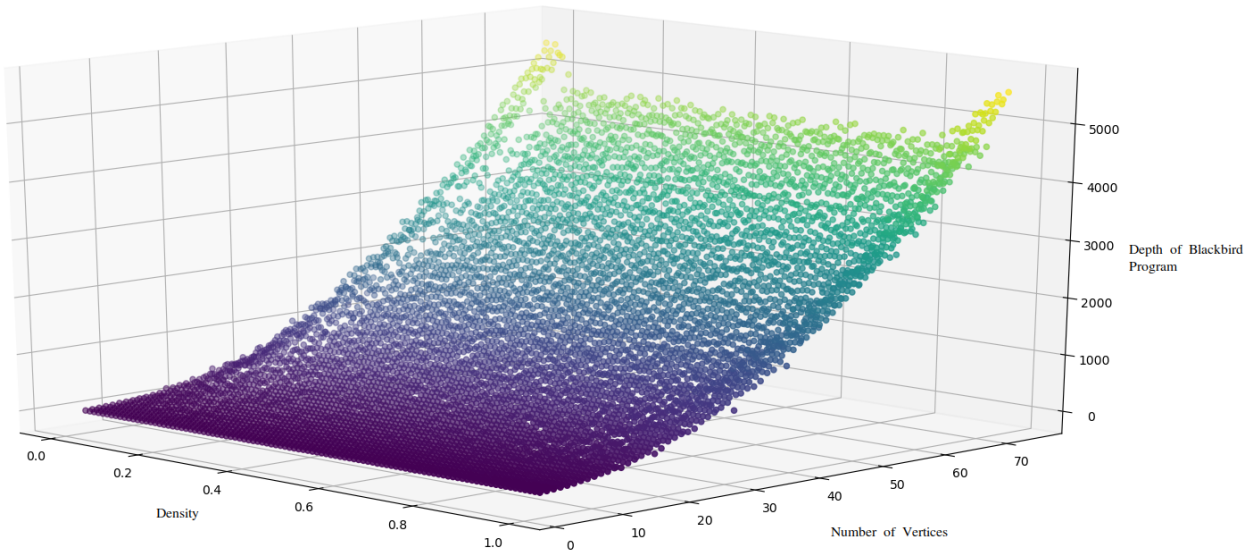


Figure 4.2: Depth of the circuit required to sample dense subgraphs vs. size of the graph vs. density of the graph

Using a density of 0.5, the depth of the graph was compared to the plot of $|V|^2$ scaled at 0.8 and 0.85. As one can see, the plot shows an upper bound of $0.85 * |V|^2$ and a lower bound of $0.8 * |V|^2$. From these results, one can reasonably assume that the depth of the quantum program to sample dense subgraphs scales with $O(|V|^2)$.

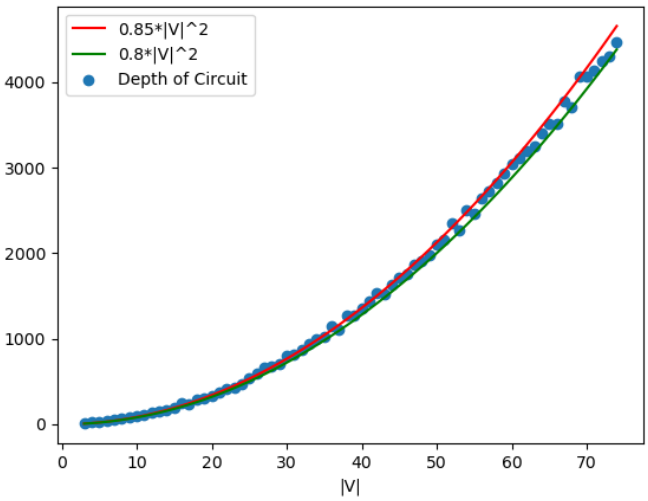


Figure 4.3: Depth of the circuit required to sample dense subgraphs vs. size of the graph, compared to a scaled plot of $|V|^2$

Chapter 5

Comparison

5.1 Algorithm Comparisons

Given that the two QC models are incredibly different, it makes sense that their algorithms to solve the Maximum Clique Problem are completely different. While the QB model is still imperfect and requires multiple executions to ensure that the correct values were measured, the number of executions required to get the expected answer is relatively small compared to the number of subgraphs that must be checked in the brute force algorithm. For this solution to the maximum clique problem, Grover's algorithm was used as the backbone to the problem. Almost all of the work required to create the algorithm went into defining the oracle for Grover's algorithm.

On the other hand, the BS model is designed around sampling. The model relies on programming the probability distribution and sampling from this. To find dense subgraphs with BS, the probability distribution needs to have a higher probability of sampling bosons that coincide with dense subgraphs than with sparse subgraphs. This was done by arranging the two mode squeezers, beamsplitters, and rotation gates according to the Autonne-Takagi factorization of the graph's adjacency matrix [25]. In this case, there was a clever association between the number of perfect matchings and the Hafnian of an adjacency matrix, the number of perfect matchings in a subgraph, and the density of a subgraph.

5.2 Application Development Comparisons

As illustrated by the solutions to the maximum clique problem, the QB and BS models are incredibly different. While there are many algorithms that can be used as the backbone for solving problems using the QB model, Grover's algorithm is one of the most common and is the only one used to solve the maximum clique problem in this thesis. These are the steps used to solve the maximum clique problem and can be used to solve more problems with Grover's algorithm:

1. Determine the variables of interest. For graph problems, these are usually the vertices or the edges.

   E.g. for the maximum clique problem, the vertices are the variables of interest

   E.g. for the travelling salesman problem, the edges are the variables of interest.

2. Put these variables of interest in superposition. This means that the answer solution will individually either include or exclude the variables of interest.

3. Define the logical conditions that make an answer a solution state.

   E.g. for the maximum clique problem, the conditions are (1) the subgraph must be a clique and (2) the subgraph must have $|V| \geq k$.

4. Use quantum gates to define the logical conditions in a quantum circuit. Put the result into a target qubit. It is helpful to think of the Toffoli gates as a logical AND, the X gate as a logical NOT, and the CNOT gate as a controlled logical NOT (XOR).

5. The rest of the circuit must now be defined. The extra necessary items are putting the target qubit into the $|-\rangle$ state, the previous step must be included

and also reversed, and the diffuser must be added. If desired, the solution state can also be measured and recorded.

Application development with BS is newer and still less defined than application development with QB. From the list of applications currently well-understood [8], the dense subgraph identification, maximum clique, point processes, and vibronic spectra problems all follow these general steps:

1. Find a connection between a probability distribution that can be sampled by bosons and the problem.

   E.g. for dense subgraph identification, the probability distribution of the Hafnian of the adjacency matrix was arranged such that the densest subgraphs were most likely to be sampled.

   E.g. the maximum clique problem used the dense subgraph algorithm to classically refine the samples into a desired solution.

   E.g. the point processes are sampling from the Hafnian distribution that bosons naturally occur in.

   E.g. the vibronic spectra solution mimics a molecule's interaction with light.

2. Set up the BS gates such that they create the desired probability distribution.

3. Sample from this setup. To get more consistent results, sample from the setup multiple times.

4. Perform any post-processing needed.

   E.g. to get the maximum cliques from the dense subgraphs, the samples have to undergo the shrinking and growing process defined in Section 2.2.2.

While these steps are not as general as the steps for application development using the QB model, these steps show that the BS model is not limited to solving only a few problems.

Since quantum computing is a relatively young field, there are few resources available to help computer scientists and physicists solve problems using quantum algorithms. Grover's algorithm is a well-known algorithm, but developing application specific algorithms is still uncommon. By providing an outline of how to design oracles for problems in which Grover's algorithm can solve them, this document makes it easier for future developers to design new oracles without having to learn too much about the physics behind Grover's algorithm. Currently, BS is a much younger model of quantum computing and much less research has gone into this model. By providing this outline of application development for BS, this document makes it easier for future developers to design new algorithms based off of this outline.

Chapter 6

Conclusion

In this work, a novel algorithm to the maximum clique problem using the qubit-based model was presented and compared to the boson sampling model's algorithm. The novel algorithm allows the maximum clique to be exactly solved in $O(\log_2(|V|)*\sqrt{2^{|V|}})$ time complexity instead of the current best exact solution to the maximum clique problem, in classical computing, which has a time complexity of $O(2^{|V|})$. **For researchers interested in the potential of quantum computers as accelerators, it is important to gain understanding on the different quantum approaches, their best application matches, and their application development process. This thesis' goal was to aid with this understanding through the comparison of the maximum clique problem solution on two different quantum computing approaches: gate-based and boson sampling.**

The new algorithm on the gate-based implementation will be able to find maximum cliques for graphs with many more vertices before becoming intractable, once quantum computers grow in qubit numbers. although it was not this proposal thesis to show better performance than classical computing, the scalability of the problem was studied. Quantum computers are rapidly increasing in size and in the near future will be able to solve this problem more quickly than classical computers. The novel algorithm was simulated and produced the expected results for both three vertex graphs and four vertex graphs.

This thesis also proposed a new approach to compare two values using the shuffling conjecture, resulting in a more efficient implementation in terms of qubits. The reduction in the number of qubits made it possible to simulate the quantum circuit for four node graphs, while with the traditional approach, this simulation was not possible.

The research presented can be expanded upon in numerous ways. The shuffling conjecture is not limited to use in the maximum clique algorithm and undoubtedly has uses in other algorithms, given the probabilistic nature of quantum computing. Another reason the shuffling conjecture is useful is that it can be used to reduce the number of qubits required to solve some problems. This is especially useful in the NISQ era. Bojić mentions a potential speedup for the QG algorithm in which k is updated logarithmically instead of linearly, similar to binary search compared to linear search [9]. This reduces the complexity of the QG algorithm to $O(\log_2(|V|) * \sqrt{2^{|V|}})$. Implementing this change would be a valuable improvement to the current algorithm. Furthermore, while one case study comparison between algorithms in the qubit-based model and the boson sampling model is useful, more case studies would allow computer scientists to have a better grasp on these relatively new technologies.

Bibliography

[1] "Executing programs on x8 devices," https://strawberryfields.ai/photonics/demos/tutorial$_X$8.*html*, *accessed* : 2021 − 03 − 25.

[2] C. Eltschka, F. Huber, O. Gühne, and J. Siewert, "Exponentially many entanglement and correlation constraints for multipartite quantum states," *Phys. Rev. A*, vol. 98, p. 052317, Nov 2018. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.98.052317

[3] A. M. G. S. e. a. Johnson, M., "Quantum annealing with manufactured spins," May 2011. [Online]. Available: https://doi.org/10.1038/nature10012

[4] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018. [Online]. Available: https://doi.org/10.22331/q-2018-08-06-79

[5] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134.

[6] S. J. Solenov D, Brieler J, "The potential of quantum computing and machine learning to advance clinical research and change the practice of medicine," vol. 115(5), pp. 463–467, 2018.

[7] K. Srinivasan, S. Satyajit, B. Behera, and P. Panigrahi, "Efficient quantum algorithm for solving travelling salesman problem: An ibm quantum experience," 05 2018.

[8] T. R. Bromley, J. M. Arrazola, S. Jahangiri, J. Izaac, N. Quesada, A. D. Gran, M. Schuld, J. Swinarton, Z. Zabaneh, and N. Killoran, "Applications of near-term photonic quantum computers: software and algorithms," *Quantum Science and Technology*, vol. 5, no. 3, p. 034010, May 2020. [Online]. Available: http://dx.doi.org/10.1088/2058-9565/ab8504

[9] A. Bojić, "Quantum algorithm for finding a maximum clique in an undirected graph," *Journal of Information and Organizational Sciences*, vol. 36, pp. 91–98, 12 2012.

[10] S. S. Hegde, J. Zhang, and D. Suter, "Efficient quantum gates for individual nuclear spin qubits by indirect control," *Physical Review Letters*, vol. 124, no. 22, Jun 2020. [Online]. Available: http://dx.doi.org/10.1103/PhysRevLett.124.220501

[11] E. T. from the arXiv, "The phosphorous atom quantum computing machine," May 2013. [Online]. Available: https://www.technologyreview.com/2013/05/22/178359/the-phosphorous-atom-quantum-computing-machine/

[12] A. K. B. R. e. a. Arute, F., "Quantum supremacy using a programmable superconducting processor," *Nature*. [Online]. Available: https://doi.org/10.1038/s41586-019-1666-5

[13] Qiskit, "Single qubit gates," 2020. [Online]. Available: https://qiskit.org/textbook/ch-states/single-qubit-gates.html

[14] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 212–219. [Online]. Available: https://doi.org/10.1145/237814.237866

[15] S. Aaronson and A. Arkhipov, "The computational complexity of linear optics," 2010.

[16] U. Feige, G. Kortsarz, and D. Peleg, "The dense k-subgraph problem," *Algorithmica*, vol. 29, p. 2001, 1999.

[17] J. M. Arrazola and T. R. Bromley, "Using gaussian boson sampling to find dense subgraphs," *Physical Review Letters*, vol. 121, no. 3, Jul 2018. [Online]. Available: http://dx.doi.org/10.1103/PhysRevLett.121.030503

[18] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983. [Online]. Available: https://science.sciencemag.org/content/220/4598/671

[19] X. Geng, J. Xu, J. Xiao, and L. Pan, "A simple simulated annealing algorithm for the maximum clique problem," *Information Sciences*, vol. 177, no. 22, pp. 5064–5071, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025507002988

[20] L. Banchi, M. Fingerhuth, T. Babej, C. Ing, and J. M. Arrazola, "Molecular docking with gaussian boson sampling," *Science Advances*, vol. 6, no. 23, p. eaax1950, Jun 2020. [Online]. Available: http://dx.doi.org/10.1126/sciadv.aax1950

[21] G. Chapuis, H. N. Djidjev, G. Hahn, and G. Rizk, "Finding maximum cliques on the d-wave quantum annealer," 2018.

[22] "qiskit.circuit.library.integercomparator," https://qiskit.org/documentation/stubs/qiskit.circuit.library.IntegerComparator.html, accessed: 2021-03-18.

[23] "The atoms of computation," https://qiskit.org/textbook/ch-states/atoms-computation.html, accessed: 2021-03-18.

[24] "Blackbird programming language," https://strawberryfields.ai/photonics/demos/run_blackbird.html, accessed: 2021-03-25.

[25] G. Cariolaro and G. Pierobon, "Bloch-messiah reduction of gaussian unitaries by takagi factorization," *Phys. Rev. A*, vol. 94, p. 062109, Dec 2016. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.94.062109