Quantum Algorithm Implementations for Beginners

Patrick J. Coles, Stephan Eidenbenz,* Scott Pakin, Adetokunbo Adedoyin, John Ambrosiano, Petr Anisimov, William Casper, Gopinath Chennupati, Carleton Coffrin, Hristo Djidjev, David Gunter, Satish Karra, Nathan Lemons, Shizeng Lin, Andrey Lokhov, Alexander Malyzhenkov, David Mascarenas, Susan Mniszewski, Balu Nadiga, Dan O'Malley, Diane Oyen, Lakshman Prasad, Randy Roberts, Phil Romero, Nandakishore Santhi, Nikolai Sinitsyn, Pieter Swart, Marc Vuffray, Jim Wendelberger, Boram Yoon, Richard Zamora, and Wei Zhu

Los Alamos National Laboratory, Los Alamos, New Mexico, USA

As quantum computers have become available to the general public, the need has arisen to train a cohort of quantum programmers, many of whom have been developing classic computer programs for most of their career. While currently available quantum computers have less than 100 qubits, quantum computer hardware is widely expected to grow in terms of qubit counts, quality, and connectivity. Our article aims to explain the principles of quantum programming, which are quite different from classical programming, with straight-forward algebra that makes understanding the underlying quantum mechanics optional (but still fascinating). We give an introduction to quantum computing algorithms and their implementation on real quantum hardware. We survey 20 different quantum algorithms, attempting to describe each in a succintc and self-contained fashion; we show how they are implemented on IBM's quantum computer; and in each case we discuss the results of the implementation with respect to differences of the simulator and the actual hardware runs. This article introduces computer scientists and engineers to quantum algorithms and provides a blueprint for their implementations.

Contents

*Corresponding author: eidenben@lanl.gov; LA-UR:2018-22975

I. INTRODUCTION

Quantum computing exploits quantum-mechanical effects—in particular superposition, entanglement, and quantum tunneling—to more efficiently execute a computation. Compared to traditional, digital computing, quantum computing offers the potential to dramatically reduce both execution time and energy consumption. These potential advantages, steady advances in nano-manufacturing, and the slow-down of traditional hardware scaling laws (such as Moore's Law) have led to a substantial commercial and national-security interest and investment in quantum computing technology in the 2010s. Consequently, there is now is a fairly wide-ranging consensus that quantum supremacy—the watershed moment where a quantum computer performs a (perhaps uninteresting) calculation that would be intractable on a classical supercomputer—is imminent. (See [1] for a precise technical definition of quantum supremacy.)

While the mathematical basis of quantum computing, the programming model, and most quantum algorithms have been published decades ago (starting in the 1990s), they have been of interest only to a small dedicated community. We believe the time has come to make quantum algorithms and their implementations accessible to a broad swath of researchers and developers across computer science, software engineering, and other fields. The quantum programming model is fundamentally different from traditional computer programming. It is also dominated by physics and algebraic notations that at times present unnecessary entry barriers for mainstream computer scientists and other more mathematically trained scientists.

In this article, we provide a self-contained, succinct description of quantum computing and of the basic quantum algorithms. Since real quantum computers, such as IBM's QX [51, 53], are now available as a cloud service, we present results from simulator and actual hardware experiments for smaller input data sets. Other surveys of quantum algorithms with a different target audience and also without actual implementations include [6, 22, 57, 73, 74, 89].

A. The Quantum Computing Programming Model

A *qubit* is a two-dimensional quantum-mechanical system that is in a state

$$|\phi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\ . \tag{1}$$

The *ket-notation* $|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}$ and $|1\rangle = \begin{pmatrix}0\\1\end{pmatrix}$ is shorthand for the vectors encoding the two basis states. Qubit implementations and technologies are a very active area of research that is not the focus of our article, we refer the reader to [63] for a survey. α and β are complex numbers with $|\alpha|^2 + |\beta|^2 = 1$. If the qubit gets measured it will return the classical bit value 0 with probability $|\alpha|^2$ or bit value 1 with probability $|\beta|^2$.

A qubit or a system of qubits changes its state by going through a series of *unitary transformations*. A unitary transformation is described by a matrix U with complex entries. The matrix U is unitary if

$$UU^\dagger = U^\dagger U = I\,, \tag{2}$$

where $U^\dagger$ is the transposed, complex conjugate of U and I is the identity matrix. A qubit state $|\phi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ evolves under the action of the 2×2 matrix U according to

$$|\phi\rangle \rightarrow U\,|\phi\rangle = \begin{pmatrix}U_{00} & U_{01}\\ U_{10} & U_{11}\end{pmatrix}\begin{pmatrix}\alpha\\\beta\end{pmatrix} = \begin{pmatrix}U_{00}\alpha + U_{01}\beta\\ U_{10}\alpha + U_{11}\beta\end{pmatrix}\ . \tag{3}$$

The joint state of a system of qubits is described by the tensor product $\otimes$. For a system of, say, three qubits with each qubit in the state $|\gamma_j\rangle = \alpha_j\,|0\rangle + \beta_j\,|1\rangle$, for $j = 1, 2, 3$, the joint state is

$$\begin{aligned}|\gamma_1\gamma_2\gamma_3\rangle &= |\gamma_1\rangle \otimes |\gamma_2\rangle \otimes |\gamma_3\rangle &(4)\\ &= \alpha_1\alpha_2\alpha_3\,|000\rangle + \alpha_1\alpha_2\beta_3\,|001\rangle + \alpha_1\beta_2\alpha_3\,|010\rangle + \alpha_1\beta_2\beta_3\,|011\rangle\\ &\quad + \beta_1\alpha_2\alpha_3\,|100\rangle + \beta_1\alpha_2\beta_3\,|101\rangle + \beta_1\beta_2\alpha_3\,|110\rangle + \beta_1\beta_2\beta_3\,|111\rangle &(5)\end{aligned}$$

A measurement of all three qubits could result in any of the eight possibilities associated with the eight basis vectors. One can see from this example that the state space grows exponentially in the number of qubits n and that in general the number of basis vectors is 2^n.

By analogy to classical gates like NOT and AND, the basic operation on a qubit is called a *gate*, which mathematically is a unitary transformation U. In contrast to classical gates, unitaries are reversible and hence the number of input qubits always equals the number of output qubits. The most common gates are described in Table I. The X gate is the quantum version of the NOT gate. The CNOT or "controlled NOT" negates a target bit if and only if a control bit is 1. The Toffoli gate or "controlled-controlled NOT" is essentially the quantum (reversible) version of the AND gate. It negates a target bit if and only if both control bits are 1.

A set of gates that together can execute all possible quantum computations is called a *universal gate set*. Taken together, the set of all unary (i.e., acting on one qubit) gates and the binary (i.e., acting on two qubits) CNOT gate form a universal gate set. More economically, the set $\{H, T, \text{CNOT}\}$ forms a universal set. Also, the Toffoli gate by itself is universal.

A quantum algorithm can be thought of as three steps: (1) encoding of the data, which could be classical or quantum, into the state of a set of input qubits, (2) a sequence of quantum gates applied to this set of input qubits, and finally (3) measurements of one or more of the qubits at the end to obtain a classically interpretable result.

One-qubit gates	Multi-qubit gates
$H = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix}$	$\text{CNOT} = C_X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$	$C_Z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$
$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$	$\text{Controlled-}U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & U_{00} & U_{01} \\ 0 & 0 & U_{10} & U_{11} \end{pmatrix}$
$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$\text{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$\text{Toffoli} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$
$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	
$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	

TABLE I: Commonly used quantum gates.

The two quantum mechanical effects that quantum algorithms can exploit to outperform classical algorithms are *superposition* of states and *entanglement*. Superposition refers to the fact that before measurement, each qubit is indeed in both basis states simultaneously with probability proportional to the amplitude of the prefactor. Entanglement is a type of correlation that has no classical analog. For example, when two spinning particles are entangled their spins are correlated not only in one direction but in all directions. Because it has no classical analog, entanglement is an effect that is not easily simulated classically and hence is an effect that could create a quantum advantage (i.e., where quantum computers perform better than classical computers). The main challenge in quantum algorithm design is to leverage these two quantum effects.

B. Implementations on a real quantum computer

In this article, we consider IBM's publicly available quantum computers. In most cases, we specifically consider the *ibmqx4*, which is a 5-qubit computer, although in some cases we also consider the *ibmqx5*, which is a 16-qubit computer. There are several issues to consider when implementing an algorithm on real quantum computers, for example:

1. What is the available gate set with which the user can state their algorithm?
2. What physical gates are actually implemented?
3. What is the qubit connectivity (i.e., which pairs of qubits can two-qubit gates be applied to)?
4. What are the sources of noise (i.e., errors)?

We first discuss the available gate set. In IBM's graphical interface to their *ibmqx4*, the available gates are:

$$\{I, X, Y, Z, H, S, S^\dagger, T, T^\dagger, U_1(\lambda), U_2(\lambda,\phi), U_3(\lambda,\phi,\theta), \text{CNOT}\}\,. \tag{6}$$

Notice that CNOT is the only 2-qubit gate in this set; all the other gates act on one qubit. Most of these gates appear in our Table I. The gates $U_1(\lambda)$, $U_2(\lambda,\phi)$, and $U_3(\lambda,\phi,\theta)$ are continuously parameterized gates, defined as follows:

$$U_1(\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix}, \quad U_2(\lambda,\phi) = \begin{pmatrix} 1/\sqrt{2} & -e^{i\lambda}/\sqrt{2} \\ e^{i\phi}/\sqrt{2} & e^{i(\lambda+\phi)}/\sqrt{2} \end{pmatrix}, \quad U_3(\lambda,\phi,\theta) = \begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i(\lambda+\phi)}\cos(\theta/2) \end{pmatrix}. \tag{7}$$

Note that $U_3(\lambda,\phi,\theta)$ is essentially an arbitrary one-qubit gate.

The gates listed in Eq. (6) are provided by IBM for the user's convenience. However these are not the gates that are physically implemented by their quantum computer. IBM has a compiler that translates the gates in (6) into products of gates from a physical gate set. The physical gate set employed by IBM is essentially composed of three gates:

$$\{U_1(\lambda), R_X(\pi/2), \text{CNOT}\}\,. \tag{8}$$

Here, $R_X(\pi/2)$ is a rotation by angle $\pi/2$ of the qubit about it's X-axis, corresponding to a matrix similar to the Hadamard:

$$R_X(\pi/2) = \begin{pmatrix} 1/\sqrt{2} & -i/\sqrt{2} \\ -i/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}. \tag{9}$$

The reason why it could be important to know the physical gate set is that some user-programmed gates may need to be decomposed into multiple physical gates, and hence could lead to a longer physical algorithm. For example, the X gate gets decomposed into three gates: two $R_X(\pi/2)$ gates sandwiching one $U_1(\lambda)$ gate.

The connectivity of the computer is another important issue. Textbook algorithms are typically written for a fully-connected hardware, which means that one can apply a two-qubit gate to any two qubits. In practice, real quantum computers may not have full connectivity. In the *ibmqx4*, which has 5 qubits, there are 6 connections, i.e., there are only 6 pairs of qubits to which a CNOT gate can be applied. In contrast a fully connected 5-qubit system would allow a CNOT to be applied to 20 different qubit pairs. In this sense, there are 14 "missing connections". Fortunately, there are ways to effectively generate connections through clever gate sequences. For example, a CNOT gate with qubit j as the control and qubit k as the target can be reversed (such that j is the target and k is the control) by applying Hadamard gates on each qubit both before and after the CNOT, i.e.,

$$\text{CNOT}_{kj} = (H \otimes H)\text{CNOT}_{jk}(H \otimes H)\,. \tag{10}$$

Similarly, there exists a gate sequence to make a CNOT between qubits j and l if one has connections between j and k, and k and l, as follows:

$$\text{CNOT}_{jl} = \text{CNOT}_{kl}\text{CNOT}_{jk}\text{CNOT}_{kl}\text{CNOT}_{jk}\,. \tag{11}$$

Hence, using (10) and (11), one can make up for lack of connectivity at the expense of using extra gates.

Finally, when implementing a quantum algorithm it is important to consider the sources of noise in the computer. The two main sources of noise are typically gate infidelity and decoherence. Gate infidelity refers to the fact that the user-specified gate does not precisely correspond to the physically implemented gate. Gate infidelity is usually worse for multi-qubit gates than for one-qubit gates, so typically one wants to minimize the number of multi-qubit gates in one's algorithm.

Class	Problem/Algorithm	Paradigms used	Hardware	Simulation Match
Inverse Function Computation	Grover's Algorithm	GO	QX4	med
	Bernstein-Vazirani	n.a.	QX4, QX5	high
Number-theoretic Applications	Shor's Factoring Algorithm	QFT	QX4	med
Algebraic Applications	Linear Systems	HHL	QX4	low
	Matrix Element Group Representations	QFT	QX4	low
	Matrix Product Verification	GO	QX4	high
	Subgroup Isomorphism	QFT	none	n.a.
	Persistent Homology	GO, QFT	QX4	med-low
Graph Applications	Graph Properties Verification	GO	QX4	med
	Minimum Spanning Tree	GO	QX4	med-low
	Maximum Flow	GO	QX4	med-low
	Approximate Quantum Algorithms	SIM	QX4	high
Learning Applications	Quantum Principal Component Analysis (PCA)	QFT	QX4	med
	Quantum Support Vector Machines (SVM)	QFT	none	n.a.
	Partition Function	QFT	QX4	med-low
Quantum Simulation	Schroedinger Equation Simulation	SIM	QX4	low
	Transverse Ising Model Simulation	VQE	none	n.a.
Quantum Utilities	State Preparation	n.a.	QX4	med
	Quantum Tomography	n.a.	QX4	med
	Quantum Error Correction	n.a.	QX4	med

TABLE II: Overview of studied quantum algorithms. Paradigms include Grover Operator (GO), Quantum Fourier Transform (QFT), Harrow/Hassidim/Lloyd (HHL), Variational Quantum Eigenvalue solver (VQE), and direct Hamiltonian simulation (SIM). The simulation match column indicates how well the hardware quantum results matched the simulator results

Decoherence refers to the fact that gradually over time the quantum computer loses its "quantumness" and becomes more like a classical object. After decoherence has fully occurred, the computer can no longer take advantage of quantum effects. This introduces progressively more noise as the quantum algorithm proceeds in time. Ultimately this limits the depth of quantum algorithms that can be implemented on quantum computers. It is worth noting that different qubits decohere at different rates, and one can use this information to better design one's algorithm. On *ibmqx4*, qubits q1 and q3 respectively have long and short decoherence times, so one might try to avoid q3 and instead use q1.

C. Quantum Algorithm Classes

Quantum algorithms are often grouped into number-theory-based, oracle-based, and quantum simulation algorithms, such as for instance on the excellent Quantum Zoo site [57], which is largely based on the main quantum algorithmic paradigm that these algorithms use. These paradigms are the Quantum Fourier Transform (QFT), the Grover Operator (GO), the Harrow/Hassidim/Lloyd (HHL) method for linear systems, variational quantum eigenvalue solver (VQE), and direct Hamiltonian simulation (SIM). The fact that most known quantum algorithms are based on these few paradigms in combination is remarkable and perhaps surprising. The discovery of additional quantum algorithm paradigms, which should be the subject of intense research, could make quantum algorithms applicable across a much wider range of applications.

We choose instead to group quantum algorithms in application areas: inverse function computation, number-theory applications, algebraic applications, graph applications, learning applications, quantum simulation, and utilities. Table II shows the algorithms studied in this paper, classified according to their application.

D. Outline

The structure of the rest of the paper is to present each of the algorithms shown in Table II, one after the other. In each case, we first discuss the goal of the algorithm (the problem it attempts to solve). Then we describe the gate

sequence required to implement this algorithm. Finally, we show the results from implementing this algorithm on IBM's quantum computer.

II. GROVER'S ALGORITHM

A. Problem definition and background

Grover's algorithm as initially described [48] enables one to find (with probability $> 1/2$) a specific item within a randomly ordered database of N items using $O(\sqrt{N})$ operations. By contrast, a classical computer would require $O(N)$ operations to achieve this. Therefore, Grover's algorithm provides a quadratic speedup over an optimal classical algorithm. It has also been shown [9] that Grover's algorithm is optimal in the sense that no quantum Turing machine can do this in less than $O(\sqrt{N})$ operations.

While Grover's algorithm is commonly thought of as being useful for searching a database, the basic ideas that comprise this algorithm are applicable in a much broader context. This approach can be used to accelerate search algorithms where a "quantum oracle" can be constructed that distinguishes the needle from the haystack. The needle and hay need not be part of a database. For example, it could be used to search to two integers $1 < a < b$ such that $ab = n$ for some number n, resulting in a factoring algorithm Of course, the performance of Grover's algorithm would not match the performance of Shor's algorithm [93, 95] for this purpose.

Implementing the quantum oracle can be reduced to constructing a quantum circuit that flips an ancillary qubit, q, if a function, $f(\mathbf{x})$, evaluates to 1 for an input $\mathbf{x}$. The function $f(\mathbf{x})$ is defined by

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} = \mathbf{x}^* \\ 0 & \text{if } \mathbf{x} \neq \mathbf{x}^* \end{cases} \tag{12}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ are binary variables and $\mathbf{x}^*$ is the item that is being sought. It may seem paradoxical at first that an algorithm for finding $\mathbf{x}^*$ is needed if such a function can be constructed. The key here is that $f(\mathbf{x})$ need only recognize $\mathbf{x}^*$ – it is similar to the difference between writing down an equation and solving an equation. For example, it is easy to check if the product of a and b is equal to n, but harder to factor n.

Here we implement a simple instance of Grover's algorithm. That is, the quantum oracle we utilize is a very simple one. Let $\mathbf{x} = (x_1, x_2)$ and we wish to find $\mathbf{x}^*$ such that $x_1^* = 1$ and $x_2^* = 1$. While finding such an x^* is trivial, we don a veil of ignorance and proceed as if it were not. This essentially means that our function $f(\mathbf{x})$ is an AND gate. Searching for an AND gate in the set of quantum gates will lead to disappointment. Since an AND gate is not reversible, it cannot be part of a quantum circuit. However, a Toffoli gate [104] is similar to a classical AND gate, but comes with an extra feature that is useful for our purposes. The Toffoli gate takes three bits as input and outputs three bits. The first two bits are unmodified. The third bit is flipped if the first two bits are 1. In other words, the Toffoli gate implements our desired quantum oracle where the first two inputs are x_1 and x_2 and the third bit is the ancillary bit, q. The behavior of the oracle in general is $|\mathbf{x}\rangle |q\rangle \rightarrow |\mathbf{x}\rangle |f(\mathbf{x}) \bigoplus q\rangle$.

B. Algorithm description

Here we present a brief introduction to Grover's algorithm. A more detailed account can be found in Nielsen and Chuang [75]. An operator, called the Grover operator, is they key piece of machinery in Grover's algorithm. This operator is defined by

$$G = (2 |\psi\rangle \langle\psi| - I)O \tag{13}$$

where ψ is the uniform superposition of all states and O is the oracle operator (see Fig. 1 for a concrete representation of this operator in the case where $\mathbf{x}$ consists of 2 bits). The action of $(2 |\psi\rangle \langle\psi| - I)$ on an arbitrary state is

$$(2 |\psi\rangle \langle\psi| - I) \sum_i a_i |i\rangle = \sum_i (2 \langle a\rangle - a_i) |i\rangle \tag{14}$$

where $\langle a\rangle = \frac{\sum_i a_i}{\sum_i 1}$ and the sums are over all possible states (the sum is finite). From equation 14 one can see that the amplitude of each state is flipped about the mean amplitude.

In order to leverage the Grover operator to successfully perform a search, $|\mathbf{x}\rangle |q\rangle$ must be appropriately initialized. The initialization is carried out by applying a Hadamard transform ($H^{\otimes n}$) on $|\mathbf{x}\rangle$ and applying a Pauli X transform

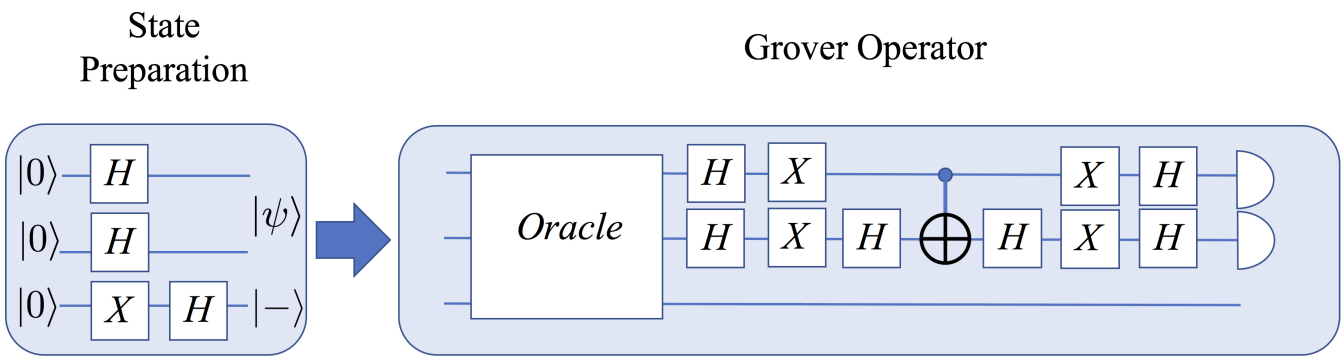


FIG. 1: A schematic diagram of Grover's algorithm is shown. Note that in this case, one application of the Grover operator is performed. This is all that is necessary when there are only two bits in $\mathbf{x}$, but the Grover operator should be applied repeatedly for larger problems.

followed by a Hadamard transform (HX) to q. This leaves $|\mathbf{x}\rangle$ in the uniform superposition of all states, $|\psi\rangle$, and $|q\rangle$ in the state $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. After performing these operations, the system is in the state $|\psi\rangle\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. Using equation 14, we can now understand how the Grover operator works. The action of the oracle operator on $|\mathbf{x}^*\rangle\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ reverses the amplitude of that state

$$O|\mathbf{x}^*\rangle\frac{|0\rangle-|1\rangle}{\sqrt{2}}\rightarrow|\mathbf{x}^*\rangle\frac{|f(\mathbf{x}^*)\bigoplus 0\rangle-|f(\mathbf{x}^*)\bigoplus 1\rangle)}{\sqrt{2}}=|\mathbf{x}^*\rangle\frac{|1\rangle-|0\rangle}{\sqrt{2}}=-|\mathbf{x}^*\rangle\frac{|0\rangle-|1\rangle}{\sqrt{2}} \tag{15}$$

A similar argument shows that all other states are unmodified by the oracle operator. Combining this with equation 14 reveals why the Grover operator is able to successfully perform a search. Consider what happens on the first iteration: The oracle operator makes it so that the amplitude of $|\mathbf{x}^*\rangle$ is below $\langle a\rangle$ (using the notation of equation 14) while all the other states have an amplitude that is slightly above $\langle a\rangle$. The effect of applying $2|\psi\rangle\langle\psi|-I$ is then to make $|\mathbf{x}^*\rangle$ have an amplitude above the mean while all other states have an amplitude below the mean. The desired behavior of the Grover operator is to increase the amplitude of $|\mathbf{x}^*\rangle$ while decreasing the amplitude of the other states. If the Grover operator is applied too many times, this will eventually stop happening. The Grover operator should be applied about $\left\lceil\frac{\pi 2^{n/2}}{4}\right\rceil$ times where n is the number of bits in $\mathbf{x}$. In the case where $\mathbf{x}$ has two bits, a single application of Grover's operator is sufficient to find $\mathbf{x}^*$ with certainty (in theory).

C. Algorithm implemented on IBM's 5-qubit computer

Fig. 2 shows the circuit that was designed to fit the ibmqx4 quantum computer. The circuit consists of state preparation (first two time slots), a Toffoli gate (the next 13 time slots), followed by the $2|\psi\rangle\langle\psi|-I$ operator (7 time slots), and measurement (the final 2 time slots). We use $q[0]$ (in the register notation from Fig. 2) as the ancillary bit, q, and $q[1]$ and $q[2]$ as x_1 and x_2. Note that the quantum computer imposes constraints on the possible source and target of CNOT gates. Some care must be taken to choose the appropriate qubits to represent each variable so that the necessary CNOT gates can be implemented. It is possible to circumvent some of the limitations in the source and target of the CNOT gates (i.e., the source and target can be reversed), but this requires increasing the depth of the circuit. Our initial implementation utilized a Toffoli gate that used such an approach, but the circuit was significantly deeper and the results were inferior to the results with the circuit in Fig. 2.

Using the simulator, this circuit produces the correct answer $\mathbf{x}=(1,1)$ every time. We executed 1,024 shots using the ibmqx4 and $\mathbf{x}=(1,1)$ was obtained 662 times with $(0,0)$, $(0,1)$, and $(1,0)$ occurring 119, 101, and 142 times respectively. This indicates that the probability of obtaining the correct answer is approximately 65%. The deviation between the simulator and the quantum computer is apparently due to the depth of the circuit combined with the approximate nature of the quantum computer. We note that our first implementation of this algorithm which used a Toffoli gate with a depth of 23 (compared to a depth of 13 here) obtained the correct answer 48% of the time.

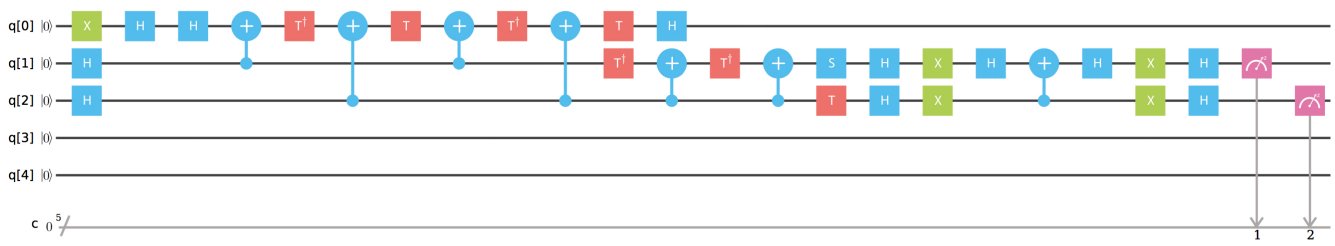


FIG. 2: The circuit that was executed on IBM's 5-qubit quantum computer (ibmqx4 was used). The first two time slots correspond to the state preparation. The next 13 time slots implement a Toffoli gate. The next 7 time slots implement the $2\left|\psi\right\rangle\left\langle\psi\right| - I$ operator, and the final two time slots are used for observing x_1 and x_2.

D. Conclusion

We designed a circuit that implements an instance of Grover's algorithm for an IBM 5-qubit quantum computer. The outcome was successful in the sense that the quantum computer successfully completed the search with a probability that is appreciably greater than 50%. However, the 65% success rate that was obtained is much lower than the 100% that is obtained by the simulator. Deeper and more complex oracles would likely produce less satisfactory results, and this is in line with our experience implementing the oracle with a deeper implementation of the Toffoli gate.

III. BERNSTEIN-VAZIRANI ALGORITHM

A. Problem definition and background

Suppose we are given a classical Boolean function, $f : \{0,1\}^n \mapsto \{0,1\}$. Such a function is guaranteed to be of the form, $f_{\mathbf{s}}(\mathbf{x}) = \langle \mathbf{s}, \mathbf{x} \rangle$. Here, $\mathbf{s}$ is an unknown vector, which we shall call a *hidden string*; and $\langle \cdot, \cdot \rangle$ is the usual vector *dot* product. Let us also suppose that we are given a black-box implementation of this function, and are tasked with identifying the hidden string $\mathbf{s}$ with the least possible number of queries to the black box.

Since each classical query to this function produces just 1 bit of information, and since an arbitrary hidden string $\mathbf{s}$ has n-bits of information, the classical *query complexity* is seen to be n. Even with bounded error, there is no way that this classical complexity can be brought down, as can be seen using slightly more rigorous information theoretic arguments.

Bernstein and Vazirani [10] built upon the earlier work of Deutsch and Jozsa [32] in theoretically exploring quantum query complexity. Their contribution to the field was a quantum algorithm for the hidden string problem, which has a non-recursive quantum query complexity of just 1. This constitutes a polynomial $\mathcal{O}(n)$ query-complexity separation between classical and quantum computation. They also discovered a less widely known recursive hidden-string query algorithm, which shows a $\mathcal{O}(n^{\log n})$ separation between classical and quantum query-complexities. These developments preceded the more famous results of Shor and Grover, and kindled a lot of early academic interest in the inherent power of quantum computers.

One thing to note with respect to the Bernstein-Vazirani (BV) quantum hidden-string query algorithm is that the *black-box* function $f_{\mathbf{s}}(\cdot)$ can be very complex to implement using reversible quantum gates—for an n-bit hidden string, the number of simple gates needed to implement $f_{\mathbf{s}}(\cdot)$ scales typically as $\mathcal{O}(4^n)$. Since the black box is a step in the algorithm, its serial execution time could in the worst-case even scale exponentially as $\mathcal{O}(4^n)$. The real breakthrough of this quantum algorithm lies in speeding up the query complexity and not the execution time per se.

B. Algorithm description

Let us explore the BV algorithm in more detail. The n-qubit Hadamard operator is defined as:

$$H^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x},\mathbf{y} \in \{0,1\}^n} (-1)^{\langle \mathbf{x},\mathbf{y} \rangle} \left|\mathbf{y}\right\rangle \left\langle \mathbf{x}\right| \tag{16}$$

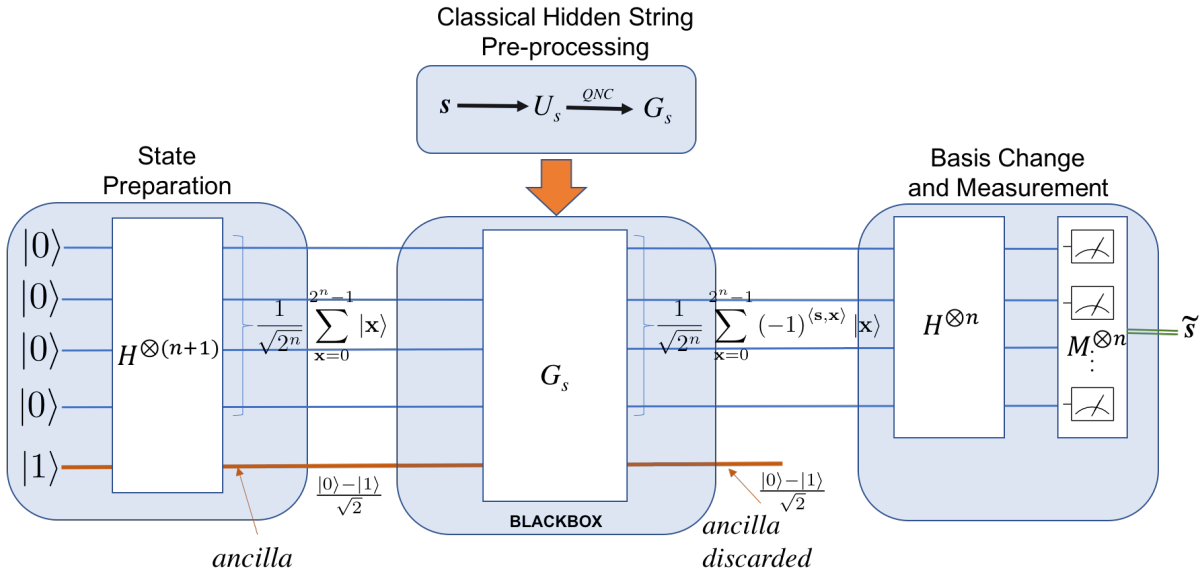


FIG. 3: Bernstein-Vazirani hidden string discovery quantum algorithm. The hidden string $\mathbf{s}$ is discovered with just a single query. The measurement result $\widetilde{\mathbf{s}}$ gives estimated hidden string.

Given a specified Boolean function $f(\cdot)$, let us introduce an $(n+1)$-qubit black-box operator:

$$U_{f(\cdot)} = \sum_{\mathbf{x}\in\{0,1\}^n, y\in\{0,1\}} |\mathbf{x}\rangle\langle\mathbf{x}| \otimes |y \oplus f(\mathbf{x})\rangle\langle y| \tag{17}$$

where we used $\oplus$ to denote Boolean (mod 2) addition; and $\otimes$ to represent Kronecker product. A nice property of this black-box operator is called *phase kickback*: when we give a particular type of input to this operator, the hidden function $f(\cdot)$ appears in the phase of the output. Denoting $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$, we get

$$U_{f(\cdot)}\left(|\mathbf{x}\rangle|-\rangle\right) = (-1)^{f(\mathbf{x})}|\mathbf{x}\rangle|-\rangle \tag{18}$$

For the BV algorithm, the Boolean function $f(\cdot)$ is just the dot product, $\langle\mathbf{s},\mathbf{x}\rangle$, where $\mathbf{s}$ is a hidden string which we want to identify using the least number of queries. So,

$$U_{\mathbf{s}} = \sum_{\mathbf{x}\in\{0,1\}^n, y\in\{0,1\}} |\mathbf{x}\rangle\langle\mathbf{x}| \otimes |y \oplus \langle\mathbf{s},\mathbf{x}\rangle\rangle\langle y| \tag{19}$$

The entire BV algorithm is represented in Figure 3. Analyzing the diagram, we see:

$$\begin{aligned} |0\rangle^n|1\rangle &\xrightarrow{H^{\otimes(n+1)}} \frac{1}{\sqrt{2^n}}\sum_{\mathbf{x}=0}^{2^n-1}|\mathbf{x}\rangle\otimes\frac{|0\rangle-|1\rangle}{\sqrt{2}} \xrightarrow{U_s} \frac{1}{\sqrt{2^n}}\sum_{\mathbf{x}=0}^{2^n-1}(-1)^{\langle\mathbf{s},\mathbf{x}\rangle}|\mathbf{x}\rangle\otimes\frac{|0\rangle-|1\rangle}{\sqrt{2}} \\ &\xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n}}\sum_{\mathbf{x},\mathbf{y}=0}^{2^n-1}(-1)^{\langle\mathbf{s},\mathbf{x}\rangle\oplus\langle\mathbf{x},\mathbf{y}\rangle}|\mathbf{y}\rangle\otimes\frac{|0\rangle-|1\rangle}{\sqrt{2}} \equiv |\mathbf{s}\rangle\otimes\frac{|0\rangle-|1\rangle}{\sqrt{2}} \end{aligned} \tag{20}$$

C. Algorithm implemented on IBM's 5-qubit and 16-qubit computers

From the BV algorithm description in the previous section, we see that in any practical implementation of this algorithm, the main ingredient is the construction of the black-box oracle $U_{\mathbf{s}}$ given a binary hidden string $\mathbf{s}$. Let us see how this is done using an example length 2 binary hidden string "01". Equation (21) below shows how the 3-qubit operator maps the $2^3 = 8$ basis vectors. The first line is the input binary vector (in the order y, x_1, x_0), and the second line is the output binary vector.

$$U_{01} = \begin{pmatrix} 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\ 000 & 101 & 010 & 111 & 100 & 001 & 110 & 011 \end{pmatrix} \tag{21}$$

This mapping is a permutation operator $U_{01} : |\mathbf{x}, y\rangle \mapsto |\mathbf{x}, \langle 01, \mathbf{x}\rangle \oplus y\rangle$; which is a special case of a unitary operator, as its matrix representation is unitary. The next task in implementation is to lower the unitary matrix operator U_{01} to primitive gates available in the quantum computer's architecture. For the IBM-QX architecture, there are just three primitive hardware gates: $\mathtt{cx}, \mathtt{u_1}(), \mathtt{u_2}(), \mathtt{u_3}()$. Here `cx` is a control-not, where only certain control, target qubit pairs are allowed by the machine's connection topology graph. With,

$$\mathtt{u}(\theta, \phi, \lambda) = \begin{pmatrix} e^{-i(\phi+\lambda)/2}\cos\theta/2 & -e^{-i(\phi-\lambda)/2}\sin\theta/2 \\ e^{i(\phi-\lambda)/2}\sin\theta/2 & e^{i(\phi+\lambda)/2}\cos\theta/2 \end{pmatrix}, \tag{22}$$

the available primitive gates $\mathtt{u_1}, \mathtt{u_2}, \mathtt{u_3}$ can be defined as, $\mathtt{u_1}(\lambda) = \mathtt{u}(0, 0, \lambda)$, $\mathtt{u_2}(\phi, \lambda) = \mathtt{u}(\pi/2, \phi, \lambda)$, $\mathtt{u_3}(\theta, \phi, \lambda) = \mathtt{u}(\theta, \phi, \lambda)$. These have approximate time costs of $\mathtt{cx} : 1001, \mathtt{u_1} : 7, \mathtt{u_2} : 104, \mathtt{u_3} : 201$ (ns) respectively on a typical existing IBM-QX architecture—the timing model used is taken from IBM's published calibration models [102] for the primitive hardware gates.

In order to lower arbitrary unitary-matrices to the primitive gates listed above, we need to first perform a unitary diagonalization of the $2^{(n+1)} \times 2^{(n+1)}$ operator-matrix using multi-qubit-controlled single-qubit unitary Given's rotation operations. Such multi-qubit-controlled single-qubit operations can be lowered using standard techniques [77] to the hardware primitive gates $\mathtt{cx}, \mathtt{u_1}(), \mathtt{u_2}(), \mathtt{u_3}()$. Even after this step we will be left with arbitrary `cx` gates. Since both `ibmqx4, ibmqx5` computers have restricted `cx` connectivity between qubits, we will need to first bring the control qubit (c) close to the target qubit (t) in the given topology (say swap qubit c to position c'), by potentially doing upto $(m-2)$ pair-wise qubit-swaps on an m-qubit computer. Then if there is a directed edge $c' \to t$, we can apply $\mathtt{cx}(c', t)$ gate directly. On the other hand, if there is an inverted directed edge $t \to c'$ on the machine topology, then we can reverse the roles of c', t by applying $H^{\otimes 2}$ operators before and after $\mathtt{cx}(t, c')$. As the overall primitive gate counts scale as $\mathcal{O}(4^n)$ for arbitrary n-qubit unitary operators, the task quickly becomes hard to do by hand. We therefore wrote a piece of software called *Quantum Netlist Compiler (QNC)* [90] for performing this complex task. QNC can do much more than convert arbitrary unitary operators to OpenQASM-2.0 circuits—it has specialized routines implemented to generate circuits to do state-preparations, permutation operators, Gray coding to reduce gate counts, mapping to physical machine-topologies, as wells as gate-merging optimizations. Applying QNC tool to the unitary matrix $U_{\mathbf{s}}$ gives us a corresponding quantum gate circuit $G_{\mathbf{s}}$ as shown in Figure 3 for a specific bit-string $\mathbf{s}$.

QNC generated black-box circuits with following gate-counts for the non-trivial 2-bit hidden-strings: "01": 36, "11": 38, "10": 37, with estimated execution time for critical path $\sim 17\mu s$ on an ideal machine with all-to-all connection topology. For the 5-qubit `ibmqx4` machine the corresponding gate-counts where: "01": 42, "11": 43, "10": 41, with estimated execution time for critical path $\sim 15\mu s$, and for the 16-qubit `ibmqx5`, they were: "01": 66, "11": 67, "10": 67, with estimated execution time for critical path $\sim 28\mu s$. In all these cases, QNC used a specialized decomposition of U_{01}, considering its permutation matrix nature, and therefore was able to reduce gate-counts by $5\times$ over the case when this special structure was ignored. Considering that the machines' observed coherence times are of the order of $\sim 60\mu s$, these QNC optimizations were crucial to the feasibility of the resulting score. The quantum score generated by QNC for U_{01} for `ibmqx4` is shown in Figure 4. A similarly prepared score for 3-bit hidden-string "111" had a gate-count of 428 in the `ibmqx4` architecture with an estimated execution time of $153\mu s$ which was well above the machines' coherence times.

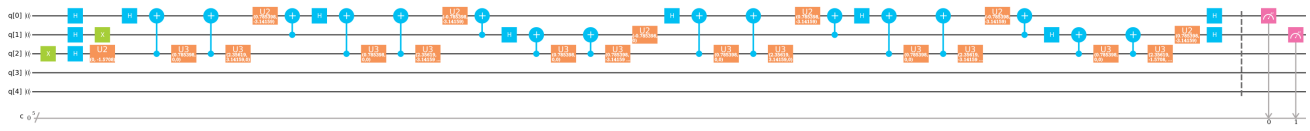


FIG. 4: Quantum score for BV algorithm with hidden string "01" targeting the `ibmqx4` architecture. There are 48 primitive gates in this score, with critical path time $\sim 16\mu s$.

We tested the QNC generated quantum scores for all non-trivial 1-qubit, 2-bit and 3-bit strings using the IBM-QISKit based local simulator. In all cases, the simulator produced the exact hidden-string as the measurement result, 100% of the trials. We then tested all 1-bit and 2-bit strings on both the 5-qubit `ibmqx4` and the 16-qubit `ibmqx5` machines. The results are shown in Figure 5. For 2-bit strings, the worst case noise was observed for the string "01" on `ibmqx4` when the qubits q_0, q_1, q_2 where used for x_0, x_1, y respectively. Since the estimated critical path times exceeded the machines' coherence times for 3-bit strings, we did not run those scores on the physical machines. Even for 2-bit strings, the scores were quite long, and the results were quite noisy even with 8192 machine-shots.

D. Conclusion

To conclude, we studied the academically important Bernstein-Vazirani quantum algorithm for hidden-string discovery. We then implemented a Quantum Netlist Compiler (QNC) to generate the black-box quantum circuits for specific IBM quantum machine topologies. We then executed the resulting scores on the simulator, as well as the 5-qubit `ibmqx4` and 16-qubit `ibmqx5` physical machines. In all cases tested, the hidden-string was identifiable from the dominant measured state provided enough number of experiments were performed. However, we also clearly observed decoherence, bias and noise issues in the physical machine results, and these issues were more pronounced as the critical path gate-count increased. As the number of primitive gates scales exponentially as the number of qubits for general unitary operators, we conclude that for successful implementation of most practical quantum algorithms, physical machines need to have large coherence times—so much so that larger coherence times are perhaps more important to have than more physical machine qubits.

IV. SHOR'S ALGORITHM FOR INTEGER FACTORIZATION

A. Problem definition and background

The integer factorization problem asks, given an integer N as an input, to find integers $1 < N_1, N_2 < N$ such that $N = N_1 N_2$. This problem is hardest when N_1 and N_2 are primes with roughly the same number of bits. If n denotes the number of bits of N, no algorithm with polynomial in n time complexity is known. The straightforward algorithm that tries all factors from 2 to $\sqrt{N}$ takes time polynomial in N, but exponential in n. The most efficient known algorithm has running time $O\left(\exp\left(\sqrt[3]{\frac{64}{9}n(\log n)^2}\right)\right)$ [82]. In practice, integers with 1000 or more bits are impossible to factor using known algorithms and classical hardware. The difficulty of factoring big numbers is the basis for the security of the RSA cryptosystem [87], one of the most widely used public-key cryptosystems.

One of the most celebrated results in quantum computing is the development of a quantum algorithm for factorization that works in time polynomial in n. This algorithm, due to Peter Shor and known as Shor's algorithm [93], runs in $O(n^3 \log n)$ time and uses $O(n^2 \log n \log\log n)$ gates. The first implementation of that algorithm on a quantum computer was reported in 2001, when the number 15 was factored [106]. The largest integer factored by Shor's algorithm so far is 21 [71].

In this section we describe Shor's algorithm and its implementation on an IBM Q experience quantum computer with 5 qubits.

B. Algorithm description

a. Reducing factorization to period finding One way to factor an integer is by using modular exponentiation. Specifically, let an odd integer $N = N_1 N_2$ be given, where $1 < N_1, N_2 < N$. Pick any integer $k < N$ such that $\gcd(k, N) = 1$, where gcd denotes the greatest common divisor. One can show that there exists an exponent $p > 0$ such that $k^p \equiv 1 \pmod{N}$. Recall that, by definition, $x \equiv y \pmod{m}$ if and only if m divides $x - y$. Assume that p is the smallest such number. If we find such p and p is even, then, by the definition of the modulo operation, N divides

$$k^p - 1 = (k^{p/2} - 1)(k^{p/2} + 1).$$

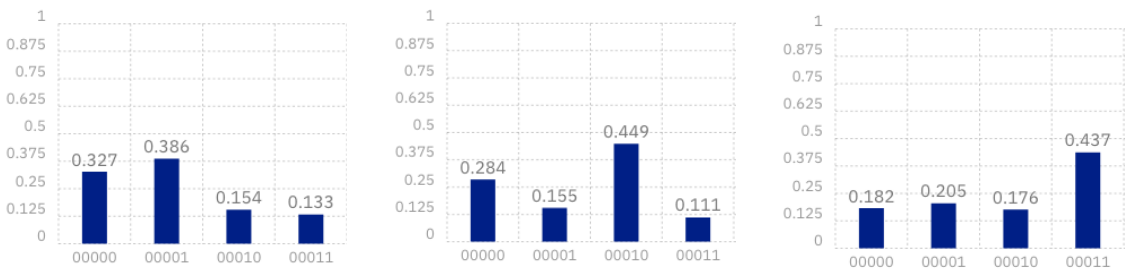


FIG. 5: Results from running the BV algorithm for 8192 shots on 2-bit hidden-strings "01", "10" and "11" respectively (left to right) on `ibmqx4`. Results for runs on `ibmqx5` were similar, but more noisy due to longer scores.

But since the difference between $n_1 = k^{p/2} + 1$ and $n_2 = k^{p/2} - 1$ is 2, n_1 and n_2 have no common factor greater than 2. Moreover, both numbers are nonzeros by the minimality of p. Since $N = N_1 N_2$ was assumed to be odd, then N_1 is a factor of either n_1 or n_2. Assume N_1 is a factor of n_1. Since N_1 is also a factor of N, then N_1 divides both n_1 and N and one can find N_1 by computing $\gcd(n_1, N)$. Hence, if one can compute such p, one can find the factors of N.

In order to find p, consider the modular exponentiation sequence $A = a_0, a_1, \ldots$, where $a_i = k^i \pmod N$. Each a_i is a number from the finite set $\{0, \ldots, N-1\}$, and hence there exists indices q and r such that $a_q = a_r$. If q and r are the smallest such indices, one can show that $q = 0$ and A is periodic with period r. For instance, for $N = 15$ and $k = 7$, the modular exponentiation sequence is $1, 7, 4, 13, 1, 7, 4, 13, 1, \ldots$ with period 4. Since the period 4 is an even number, we can apply the above idea to find

$$7^4 \bmod 15 \equiv 1 \Rightarrow 7^4 - 1 \bmod 15 \equiv 0 \Rightarrow (7^2 - 1)(7^2 + 1) \bmod 15 \equiv 0 \bmod 15 \Rightarrow 15 \text{ divides } 48 \cdot 50,$$

which can be used to compute the factors of 15 as $\gcd(48, 15) = 3$ and $\gcd(50, 15) = 5$.

Finding the period of the sequence A is, however, not classically easier than directly searching for factors of N, since one may need to check as many as $\sqrt{N}$ different values of A before encountering a repetition. However, with quantum computing, the period can be found in polynomial time using the Quantum Fourier Transform (QFT). In the next subsection we will give a definition and describe a circuit for computing QFT.

b. Quantum Fourier Transform The Discrete Fourier Transform (DFT) takes as an input a vector X of size N and outputs vector $Y = WX$ where the *Fourier matrix* W is defined by

$$W = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \ldots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \ldots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \ldots & \omega^{(N-1)(N-1)} \end{bmatrix},$$

where the ij-the element of the matrix is $W_{ij} = \omega^{ij}$ and ω is a primitive N-th root of unity ($\omega^N = 1$). A straightforward implementation of the matrix-vector multiplication takes $O(N^2)$ operations, but, by using the special structure of the matrix, the Fast Fourier Transform (FFT) does the multiplication in only $O(N \log N)$ time. The algorithm is recursive and is illustrated on Figure 6. The Quantum Fourier Transform (QFT) is defined as a transformation between two

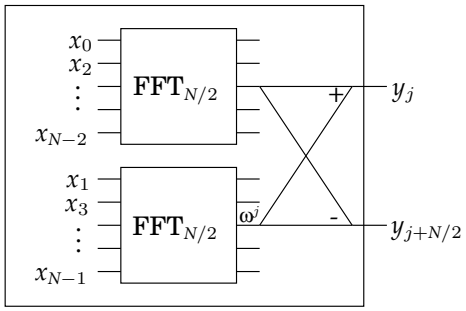


FIG. 6: Fast Fourier Transform circuit, where j denoted a row from the top half of the circuit and ω^j denotes that the corresponding value is multiplied by ω^j. The plus and minus symbols indicate that the corresponding values have to be added or subtracted, respectively.

quantum states that are determined using the values of DFT (FFT). If W is a Fourier matrix and $X = \{x_i\}$ and $Y = \{y_i\}$ are vectors such that $Y = WX$, then QFT is defined as the transformation

$$\mathcal{QFT}(\sum_{k=0}^{N-1} x_k |k\rangle) = \sum_{k=0}^{N-1} y_k |k\rangle .$$

The implementation of QFT mimics the stages (recursive calls) of FFT, but implements each stage using only $n + 1$ additional gates per stage. A single Hadamard gate on the last (least significant) bit implements the additions/subtractions of the outputs from the recursive call and the multiplications by ω^j are done using n controlled phase gates. The circuit for $n = 5$ is shown on Figure 7.

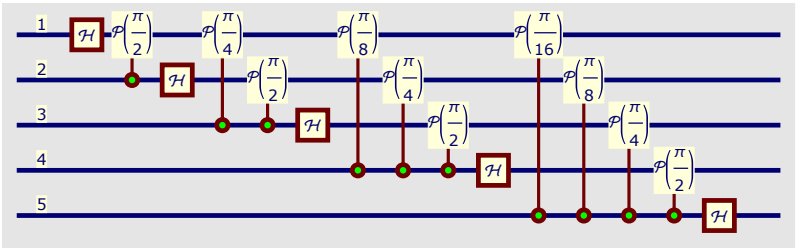


FIG. 7: A Quantum Fourier Transform circuit for five qubits ($n = 5$).

The property of QFT that is essential for the factorization algorithm is that it can "compute" the period of a periodic input. Specifically, if the input vector X is of length M and period r, where r divides M, and its elements are of the form

$$x_i = \begin{cases} \sqrt{r/M} & \text{if } i \bmod r \equiv s \\ 0 & \text{otherwise} \end{cases}$$

for some offset $s < r$, and $\mathcal{QFT}\left(\sum_{i=0}^{M} x_i \left|i\right\rangle\right) = \sum_{i=0}^{M} y_i \left|i\right\rangle$, then

$$y_i = \begin{cases} 1/\sqrt{r} & \text{if } i \bmod M/r \equiv 0 \\ 0 & \text{otherwise} \end{cases}$$

i.e., the output has nonzero values at multiples of M/r (the values $\sqrt{r/M}$ and $1/\sqrt{r}$ are used for normalization). Then, in order to factor an integer, one can find the period of the corresponding modular exponentiation sequence using QFT, if one is able to encode its period in the amplitudes of a quantum state (the input to QFT).

A period-finding circuit for solving the integer factorization problem is shown on Figure 8 [29]. The first QFT on

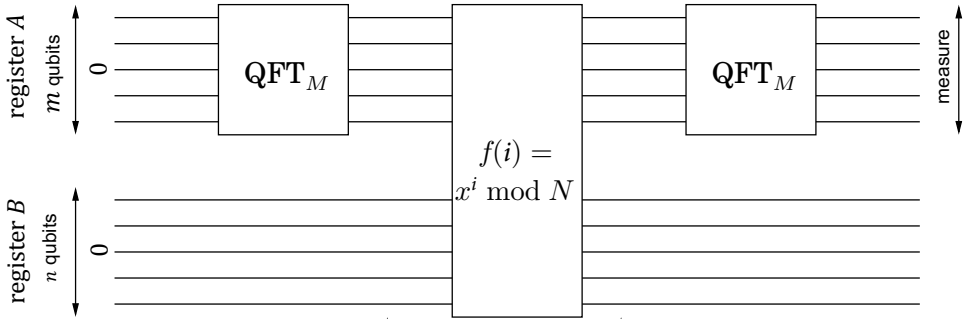


FIG. 8: Illustration of the period-finding circuit, where $m = 2n$ and $M = 2^m$.

register A produces an equal superposition of the qubits from A, i.e., the resulting state is

$$\frac{1}{\sqrt{M}} \sum_{i=0}^{M} \left|i, 0\right\rangle .$$

Next is a modular exponentiation circuit that computes the function $f(i) = x^i \pmod N$ on the second register. The resulting state is

$$\frac{1}{\sqrt{M}} \sum_{i=0}^{M} \left|i, f(i)\right\rangle .$$

Before we apply the next QFT transform, we do a measurement of register B. (By the principle of deferred measurement [75] and due to the fact that register A and B don't interact from that point on, we don't have to

actually implement the measurement, but it will help to understand the final output.) If the value measured is s, then the resulting state becomes

$$\frac{1}{\sqrt{M/r}} \sum_{\substack{i=0 \\ f(i)=s}}^{M} |i, s\rangle \,,$$

where r is the period of $f(i)$. In particular, register A is a superposition with equal non-zero amplitudes only of $|i\rangle$ for which $f(i) = s$, i.e., it is a periodic superposition with period r. Given the property of QFT, the result of the transformation is the state

$$\frac{1}{\sqrt{r}} \sum_{i=0}^{r} |i(M/r), s\rangle \,.$$

Hence, the measurement of register A will output a multiple of M/r. If the simplifying assumption that r divides M is not made, then the circuit is the same, but the classical postprocessing is a bit more involved.

C. Algorithm implemented on IBM's 5-qubit computer

We implemented the algorithm on ibmqx4, a 5-qubit quantum processor from the IBM Quantum Experience, in order to factor number 15 with $x = 11$. The circuit as described on Figure 8 requires 12 qubits and 196 gates, too large to be implemented on ibmqx4. Hence, we used an optimized/compiled version from [106] that uses 5 qubit and 11 gates (Figure 9). The results from the measurements are shown on Figure 10.

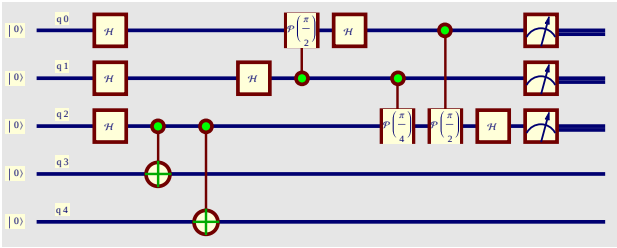


FIG. 9: Circuit for Shor's algorithm for $N = 15$ and $x = 11$.

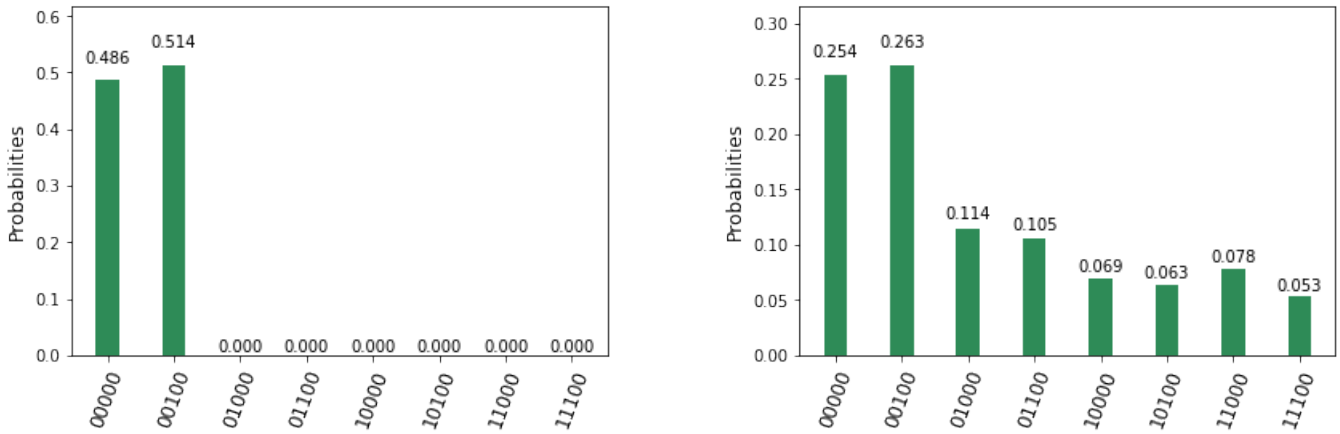


FIG. 10: Output from the circuit from Figure 9 implemented on the simulator (left) and ibmqx4 (right).

The periods found by the simulator are $p = 0$, which is ignored as a trivial period, and $p = 4$, which is a good one. Since $M = 8$, we can conclude that r divides $M/p = 8/4 = 2$, hence $r = 2$. Then 15 divides

$$(x^r - 1) = (11^2 - 1) = (11 - 1)(11 + 1) = 10 \cdot 12.$$

By computing $\gcd(15, 10) = 5$ and $\gcd(15, 12) = 3$, we find the factors of 15.

The output from ibmqx4 finds the same periods 0 and 4 with the highest probabilities, but contains much more noise.

D. Conclusion

Shor's quantum factorization algorithm reduces to finding the period of a periodic sequence, and such period can be found using Quantum Fourier Transform. Unless optimized, the general-case circuit for Shor's algorithm for factorization of the number 15 is too large, both with respect to the number of qubits as well as the number of gates, to be implemented on a 5-qubit processor. We were able to implement a compiled optimized version that produced correct results, however, comparing the results produced by the simulator and the real quantum processor showed the considerable noise of the latter.

V. LINEAR SYSTEMS

A. Problem definition and background

Solving linear systems is central to a majority of science, engineering, finance and economics applications where one comes across such systems while solving differential or partial differential equations or while performing regression, for example. The problem of solving a system of linear equations involves: Given a system $A\vec{x} = \vec{b}$, find $\vec{x}$ for a given matrix $\vec{A}$ and vector $\vec{b}$. Here we assume that A is a Hermitian matrix, in that it is self-adjoint. To represent $\vec{x}$, $\vec{b}$ as quantum states $|x\rangle$, $|b\rangle$, respectively, one has to rescale them as unit vectors, such that $\|\vec{x}\| = \|\vec{b}\| = 1$. Thus, one can pose the problem as finding $|x\rangle$ such that

$$A\,|x\rangle = |b\rangle\,, \tag{23}$$

with the solution $|x\rangle$ being

$$|x\rangle = \frac{A^{-1}\,|b\rangle}{\|A^{-1}\,|b\rangle\,\|}. \tag{24}$$

B. Algorithm description

The quantum algorithm for the linear system was first proposed by Harrow, Hassidim, Lloyd (HHL) [49] that has been implemented on various quantum computers in [7, 18, 110]. The problem of solving for $\vec{x}$ in the system $A\vec{x} = \vec{b}$ is posed as obtaining expectation value of some operator M with $\vec{x}$, $\vec{x}^\dagger M\vec{x}$, instead of directly obtaining the value of $\vec{x}$. This is particularly useful when solving on a quantum machine, since one usually obtains probabilities with respect to some measurement, typically, these operators are Pauli's operators X, Y, Z. These probabilities can then be translated to expectation values with respect to these operators. Let $\{|u_j\rangle\}$ and $\{\lambda_j\}$ be the eigenbasis and eigenvalues of A, respectively, with the eigenvalues rescaled, such that $0 < \lambda_j < 1$. Then the state $|b\rangle$, can be written as a linear combination of the eigenbasis $\{|u_j\rangle\}$, $|b\rangle = \sum_{j=1}^{N} \beta_j\,|u_j\rangle$. The goal of the HHL algorithm is to obtain $|x\rangle$ in the form $|x\rangle = \sum_{j=1}^{N} \beta_j \frac{1}{\lambda_j}\,|u_j\rangle$. By decomposing $A = R^\dagger \Lambda R$, the HHL algorithms in a nutshell involves performing a set of operations that essentially performs the three steps:

$$R^\dagger \Lambda R\,|x\rangle = |b\rangle \overset{\text{Step1}}{\Longrightarrow} \Lambda R\,|x\rangle = R\,|b\rangle \overset{\text{Step2}}{\Longrightarrow} R\,|x\rangle = \Lambda^{-1} R\,|x\rangle \overset{\text{Step3}}{\Longrightarrow} |x\rangle = R^\dagger \Lambda^{-1} R\,|x\rangle \tag{25}$$

The HHL algorithms requires three sets of qubits: a single ancilla qubit, a register of n qubits used to store the eigenvalues of A in binary format with precision up to n bits, and a memory of $O(\log(N))$ that initially stores $|b\rangle$ and eventually stores $|x\rangle$. Start with a state $|0\rangle_a\,|0\rangle_r\,|b\rangle_m$, where the subscripts a, r, m, denote the sets of ancilla, register and memory qubits, respectively. This subscript notation was used in [110], and we found it to be most useful in keeping things clear. The HHL algorithm, in general, is as follows: (1) Perform quantum phase estimation using the unitary transformation e^{iAt}, where $t$is time. Then map the eigenvalues λ_j into the register in the binary form to transform the

system from $|0\rangle_a |0\rangle_r |b\rangle_m$ to $\sum_{j=1}^{N} \beta_j |0\rangle_a |\lambda_j\rangle_r |u_j\rangle_m$. (2) Rotate the ancilla qubit $|0\rangle_a$ to $\sqrt{1-\frac{C^2}{\lambda_j^2}}|0\rangle_a + \frac{C}{\lambda_j}|1\rangle_a$ for each λ_j. This is performed through controlled rotation on the $|0\rangle_a$ ancilla qubit. The system will evolve to

$$\sum_{j=1}^{N} \beta_j \left(\sqrt{1-\frac{C^2}{\lambda_j^2}} |0\rangle_a + \frac{C}{\lambda_j} |1\rangle_a \right) |\lambda_j\rangle_r |u_j\rangle_m . \tag{26}$$

(3) Perform the reverse of Step 1. This will lead the system to

$$\sum_{j=1}^{N} \beta_j \left(\sqrt{1-\frac{C^2}{\lambda_j^2}} |0\rangle_a + \frac{C}{\lambda_j} |1\rangle_a \right) |0\rangle_r |u_j\rangle_m . \tag{27}$$

By selecting the $|1\rangle$ state in the ancilla qubit will give

$$|x\rangle \approx \sum_{j=1}^{N} C \left(\frac{\beta_j}{\lambda_j} \right) |u_j\rangle . \tag{28}$$

These three steps are equivalent to the three steps shown in Eq. (25).

C. Example problem

We implemented the HHL algorithm on a 2 × 2 system. For this, we chose $A = \begin{pmatrix} 1.5 & 0.5 \\ 0.5 & 1.5 \end{pmatrix}$. We used four qubits for solving the system – one ancilla, one memory and two register qubits. For this case, the eigenvalues of A are $\lambda_1 = 1$ and $\lambda_2 = 2$ with the eigenvectors being $\frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$ or $|-\rangle$ and $\frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ or $|+\rangle$, respectively. For this system, the three steps of the HHL algorithm, can be performed by the operations shown in Fig. (11). For the controlled rotation, we used a controlled U rotation with $\theta = \pi$ for λ_1 and $\theta = \pi/3$ for λ_2. This is done by setting $C = 1$ in the Eq. (26). Both λ and ϕ were set a zero, in these controlled U rotations. Although the composer on Quantum Experience does not have this gate, in IBM qiskit-sdk-py, we used `cu3` function for this purpose. Three cases were used for b: $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $\frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$ and $\frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$. We post selected the states with $|1\rangle$ in the ancilla qubit. The probabilities of these states are normalized such that their sum is one. Measurements with respect to $\langle X \rangle$, $\langle Y \rangle$, $\langle Z \rangle$ were performed to obtain the expectation values. QASM code was output from qiskit-sdk-py and then uploaded on IBM Quantum Experience. Figure 12 shows the equivalent composer circuit generated from QASM for the Z measurement case.

D. Simulator and ibmqx 4 results

To first test our implementation of the algorithm, we ran nine cases on the local simulator provided by qiskit-sdk-py – three b cases and three measurements with respect to the operators X, Y, Z, for each b case. The comparison between the theoretical expectation values $\langle X \rangle$, $\langle Y \rangle$, $\langle Z \rangle$ and the simulator values are shown in Table III. The simulator expectation values and the theoretical values match well. This shows that the implementation of the algorithm gives expected results. Similar expected values were also seen using the simulator on IBM Quantum Experience instead of the local simulator. We then ran on the quantum computer ibmqx4. Figure 13 shows a comparison between the simulator results and the results from the ibmqx4 with Z measurement on the circuit. As can be seen from the figure, the results from the actual run do not give the expected answer as seen in the simulator results.

E. Conclusion

We implemented a quantum algorithm to solve a 2 × 2 linear system. Studies were performed using three cases for $|b\rangle$ – $|0\rangle$, $|+\rangle$ and $|-\rangle$. For all three cases of $|b\rangle$, the local simulator and the IBM QX simulator for 5-bit ibmqx4 gave results close to the theoretical expectation values with respect to X, Y and Z. However, the actual results from ibmqx4 were incorrect. This is possibly due to decoherence since our circuit was fairly long.

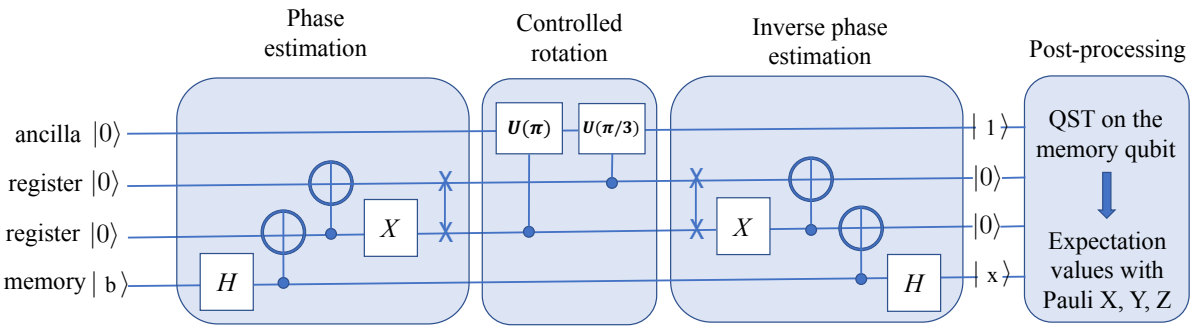


FIG. 11: Schematic of the circuit for the quantum algorithm for solving a 2 × 2 linear system. The first step involves phase estimation, which maps the eigenvalues λ_j of A into the register in the binary form. The second step involves controlled rotation of the ancilla qubit, so that the inverse of the eigenvalues $\frac{1}{\lambda_j}$ show in the state. The third step is the inverse phase estimation to disentangle the system, and restores the registers to $|0\rangle$. The memory qubit now stores $|x\rangle$, which is then post-processed to get the expectation values with respect to the Pauli operators X, Y and Z.

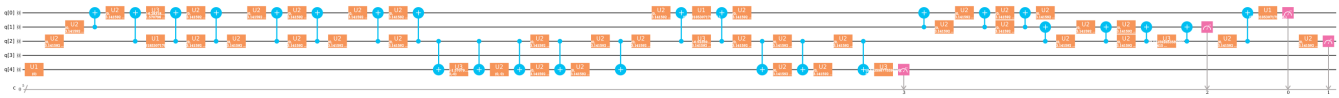


FIG. 12: Circuit implemented on IBM's 5-qubit ibmqx4 quantum computer for the case with $|b\rangle$ set to $|0\rangle$ and with $\langle Z \rangle$ measurement. After implementing the circuit in Fig. 11 and setting the coupling map of the ibmqx4 architecture, qiskit-sdk-py re-arranges the qubits to fit the mapping. This circuit was the outcome of the re-arrangement which was implemented on the ibmqx4 quantum computer.

TABLE III: Comparison between theoretical and simulator values for the expectation values $\langle X \rangle$, $\langle Y \rangle$, $\langle Z \rangle$. T stands for theoretical and S stands for simulator.

$\|b\rangle$	T $\langle X \rangle$	S $\langle X \rangle$	T $\langle Y \rangle$	S $\langle Y \rangle$	T $\langle Z \rangle$	S $\langle Z \rangle$
$\|0\rangle$	-0.6	-0.6045	0.0	-0.0271	0.8	0.8161
$\|+\rangle$	1.0	1.0	0.0	-0.0640	0.0	0.0236
$\|-\rangle$	-1.0	-1.0	0.0060	0.0	-0.0220	0.0

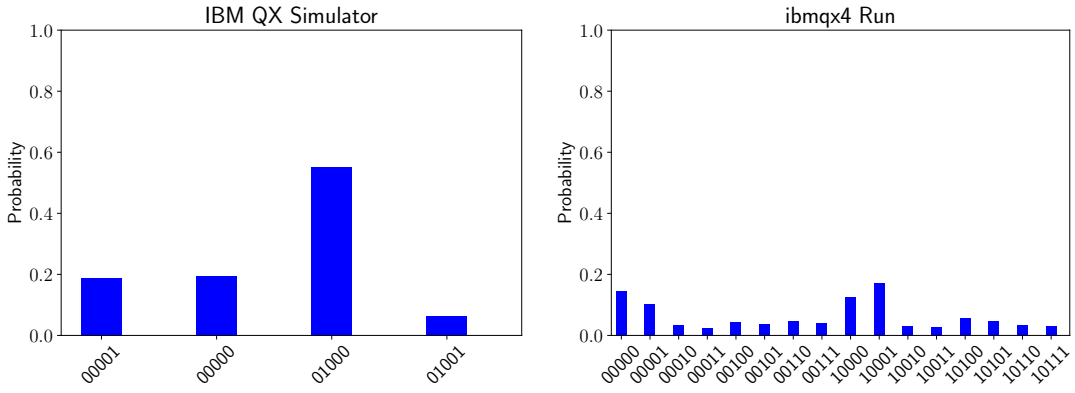


FIG. 13: Results of the circuit with Z measurement from the actual run and the simulator on a ibmqx4. 4096 shots were used for both the cases.

VI. MATRIX ELEMENTS OF GROUP REPRESENTATIONS

A. Problem definition and background

A group $(G, \cdot)$ or (G) is a mathematical object defined by its elements $(g_1, g_2, \dots)$ and an operation between elements $(\cdot)$, such that the next four properties are satisfied. (1) Closure: for any two group elements, the defined group operation produces another element, which belongs to the group (for $\forall\ g_i,g_j \in G$: $g_i \cdot g_j = g_k \in G$). (2) Associativity: for $\forall\ g_i, g_j, g_m \in G$ $g_i \cdot (g_j \cdot g_m) = (g_i \cdot g_j) \cdot g_m$. (3) Identity element: $e \in G$, such that $e \cdot g_i = g_i \cdot e = g_i$. (4) Inverse element: for $\forall\ g_i \in G$ exists g_p, such that $g_i \cdot g_p = g_p \cdot g_i = e$. A group with a finite amount of elements n is called a finite group with order n, while a group with an infinite amount of elements is an infinite group. In this note, we will discuss application of a quantum gate computer, in general, and IBM Quantum experience (5, 16-qubit chips), in particular, to solve problems related to finite groups.

Example 1A. Abelian group A_n with n elements: $0, 1, \dots, n-1$, and group operation $+\mathrm{mod}(n)$: $g_i+g_j = (i+j)\mathrm{mod}(n)$. For instance, for $n = 3$: $a_0 = 0$, $a_1 = 1$, $a_2 = 2$. Then, $a_2 + a_2 = 4\ \mathrm{mod}(3) = 1 = a_1$, $a_2 + a_1 = 3\ \mathrm{mod}(3) = 0 = a_0$, *etc.* Then identity element is $a_0 = 0$, and the inverse element is $a_i^{-1} = a_{n-i}$. This group is called Abelian or commutative, because in addition to the four group properties, it has a property of commutativity: $a_i \cdot a_j = a_j \cdot a_i$ for $\forall\ a_i,\ a_j \in A_n$.

Example 1S. Symmetry group S_n with $n!$ group elements, each is a permutation of n objects: $[1, 2.., n]$, $[2, 1.., n]$, $\dots$, $[n, n-1.., 2, 1]$. Consequent application of two permutations is a group operation. For instance, for group S_2: (e,p) we have two objects a and b. The identity element e is no permutation: $ab \to ab$, while one permutation p is the second group element: $ab \to ba$. Then, $p \cdot p = e$, and $p^{-1} = p$. Only S_1 and S_2 are Abelian groups. For $n \geq 3$ S_n are not commutative. Let us write elements of group S_3 as a permutation of elements 123 in the next order: $[123] \to [123]$, $[231]$, $[312]$, $[213]$, $[132]$, $[321]$. Then $s_4 \cdot s_2 = s_6$, while $s_2 \cdot s_4 = s_5$.

While group definition is quite simple, it is not straightforward how to operate with group elements in general, especially when defined operation between them is not trivial and/or the group order, n, is large. In this case, it is helpful to apply the theory of representation to the group. The idea is simple: if we can correspond a group of unknown objects with nontrivial operation to the group of known objects with some trivial operations, we can gain some information about unknown group. In general, we introduce a function applied to a group element: $\rho\,(g_i)$. Such function defines the group representation of G if for $\forall\ g_i,\ g_j \in G$, $\rho(g_i) * \rho(g_i) = \rho(g_i \cdot g_j)$, where $(*)$ can be a different operation from $(\cdot)$.

Example 2A. Representation of Abelian group A_n: $a_j \to \rho(a_j) = e^{i2\pi j/N}$, where an original operation $(+\mathrm{mod}(n))$ is substituted by the new operation of multiplication. Note that the group S_2 can be represented the same way as A_2.

Example 2S. Representation of group S_3: $s_j \to \rho(s_j) = 1$, where the original operation is again substituted by the new operation of multiplication. Such representation of the group S_3 is trivial, since it does not carry any information about the group, however it satisfies the definition of the group representation. Moreover, [1,1, ...] is a trivial representation for any group. Another representation of group S_3: $[1, 1, 1, -1, -1, -1] \to [s_1, s_2, \dots, s_n]$, where we use the order defined above. While it carries more information about the initial group than trivial representation, it does not imply that the group S_3 is not Abelian. One would have no luck to build a one-dimensional representation for group S_3 which would retains all its properties. The smallest equivalent representation for S_3 is two-dimensional. The multidimensional representations can be easy understood when represented by matrices.

When $\rho(g)$ is a $d_\rho \times d_\rho$ matrix, such representation is referenced as a matrix representation of the order d_ρ, while $(*)$ is the operation of matrix multiplication. All representations of finite group can be expressed as unitary matrices given an appropriate choice of basis. To prove the last fact, we introduce one more definition, the regular representation, and complete it with simple examples. We choose an element g_k from the group and apply it from the left to a set of all group elements using group operation: $[g_1, g_2, \dots, g_n] \to [g_k \cdot g_1, g_k \cdot g_2, \dots, g_k \cdot g_n]$. By the first property of the group definition each element $g_k \cdot g_j$ of the new set is still an element of the group G. Hence, such operation can be described as a permutation, $R(g_k)$, between group elements, unique for each chosen g_k: $g_k \to R(g_k)$. Such correspondence defines the regular representation of the group G. Let us introduce the vector space of group elements to add some mathematical formulas in this definition, and, at the same time, use the language commonly used in quantum mechanics and quantum computing. Each element (g_i) of the group G has a vector (ket) $|g_i\rangle$ pointing on that group element or the group element number i if the group elements were sorted out in sequence. Such vector space has the orthogonality property: $\langle g_i|g_j\rangle = \delta_{ij}$, and the identity operator is expressed as $\sum_{i=1}^{N} |g_i\rangle\langle g_i| = 1$. Hence, the regular representation is the correspondence: $g_k \to R(g_k) = \sum_{i=j}^{N} |g_k \cdot g_j\rangle\langle g_j|$. The regular representation of group element, g_k, can be expressed in matrix form by implementing identity operator on the left: $R(g_k) = \sum_{i=1}^{N}\sum_{j=1}^{N} |g_i\rangle\ \langle g_i|g_k \cdot g_j\rangle\ \langle g_j|$ or $R_{ij} = \langle g_i|g_k \cdot g_j\rangle$, where i and j point on the row and column respectively. We can prove that the regular representation

is a representation, using this notation:

$$\begin{aligned} R(g_k)\cdot R(g_m) &= \sum_{i=1}^{N}\sum_{j=1}^{N} |g_k\cdot g_i\rangle \langle g_i|g_m\cdot g_j\rangle \langle g_j| \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N} |g_k\cdot g_m\cdot g_j\rangle \langle g_i|g_m\cdot g_j\rangle \langle g_j| \\ &= \sum_{j=1}^{N} |g_k\cdot g_m\cdot g_j\rangle\langle g_j| = R(g_k\cdot g_m), \end{aligned} \tag{29}$$

where we used orthogonality: $\langle g_i|g_m\cdot g_j\rangle = 1$ only if $|g_i\rangle = |g_m\cdot g_j\rangle$ and 0 otherwise, which allowed us to swap these two states. Then, we used the same fact to calculate the sum over i.

Example 3A. Regular representation of the Abelian group A_4, where each matrix element is calculated using the result derived above $R_{ij}(a_k) = \langle a_i|a_k\cdot a_j\rangle$:

$$R(a_0) = \begin{pmatrix} 1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix},\ R(a_1) = \begin{pmatrix} 0&0&0&1\\1&0&0&0\\0&1&0&0\\0&0&1&0\end{pmatrix},\ R(a_2) = \begin{pmatrix} 0&0&1&0\\0&0&0&1\\1&0&0&0\\0&1&0&0\end{pmatrix},\ R(a_3) = \begin{pmatrix} 0&1&0&0\\0&0&1&0\\0&0&0&1\\1&0&0&0\end{pmatrix}. \tag{30}$$

Commutative property is conserved: $R(a_i)\cdot R(a_j) = R(a_j)\cdot R(a_i)$.

Example 3S. Regular representation of the group S_3, where we use the same order of permutations introduced above ([123] → [123], [231], [312], [213], [132], [321])

$$R(s_1) = \begin{pmatrix} 1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1\end{pmatrix},\ R(s_2) = \begin{pmatrix} 0&0&1&0&0&0\\1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&0&0&0&1\\0&0&0&1&0&0\\0&0&0&0&1&0\end{pmatrix},\ R(s_3) = \begin{pmatrix} 0&1&0&0&0&0\\0&0&1&0&0&0\\1&0&0&0&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1\\0&0&0&1&0&0\end{pmatrix}, \tag{31}$$

$$R(s_4) = \begin{pmatrix} 0&0&0&1&0&0\\0&0&0&0&0&1\\0&0&0&0&1&0\\1&0&0&0&0&0\\0&0&1&0&0&0\\0&1&0&0&0&0\end{pmatrix},\ R(s_5) = \begin{pmatrix} 0&0&0&0&1&0\\0&0&0&1&0&0\\0&0&0&0&0&1\\0&1&0&0&0&0\\1&0&0&0&0&0\\0&0&1&0&0&0\end{pmatrix},\ R(s_6) = \begin{pmatrix} 0&0&0&0&0&1\\0&0&0&0&1&0\\0&0&0&1&0&0\\0&0&1&0&0&0\\0&1&0&0&0&0\\1&0&0&0&0&0\end{pmatrix}. \tag{32}$$

At this point, we can finally illustrate the problem of calculating matrix elements of the group representations, which is equivalent to the problem of calculating an expectation value of an operator $\mathbf{A}$ in respect to the state $|\psi\rangle$ in quantum mechanics: $\langle\mathbf{A}\rangle = \langle\psi|\,\mathbf{A}\,|\psi\rangle$, or in matrix form: $X\cdot A\cdot X^T$, where X is a vector-row and X^T is a vector-column, and $(\cdot)$ is the standard operation of matrix multiplication. In addition, in matrix form, the operator and state should be represented in the same basis. The state is usually normalized: $X\cdot X^T = 1$.

Example 4A. Calculating matrix elements of the regular representation of the element a_2 from the Abelian group A_4 in respect to the state ψ_{13} equally pointing on the group elements a_1 and a_3 in the operator and matrix form. In operator form we find:

$$\langle\psi_{12}|\,\mathbf{a_2}\,|\psi_{12}\rangle = \frac{\langle a_1| + \langle a_3|}{\sqrt{2}}\left(\sum_{i=0}^{N-1}|a_2\cdot a_i\rangle\langle a_i|\right)\frac{|a_1\rangle + |a_3\rangle}{\sqrt{2}} = \frac{\langle a_3|a_2\cdot a_1\rangle\langle a_1|a_1\rangle}{2} + \frac{\langle a_1|a_2\cdot a_3\rangle\langle a3|a3\rangle}{2} = 1\,. \tag{33}$$

In matrix form:

$$\langle\psi_{12}|\,\mathbf{a_2}\,|\psi_{12}\rangle = \begin{pmatrix}0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}}\end{pmatrix}\cdot\begin{pmatrix}0&0&1&0\\0&0&0&1\\1&0&0&0\\0&1&0&0\end{pmatrix}\cdot\begin{pmatrix}0\\\frac{1}{\sqrt{2}}\\0\\\frac{1}{\sqrt{2}}\end{pmatrix} = 1\,. \tag{34}$$

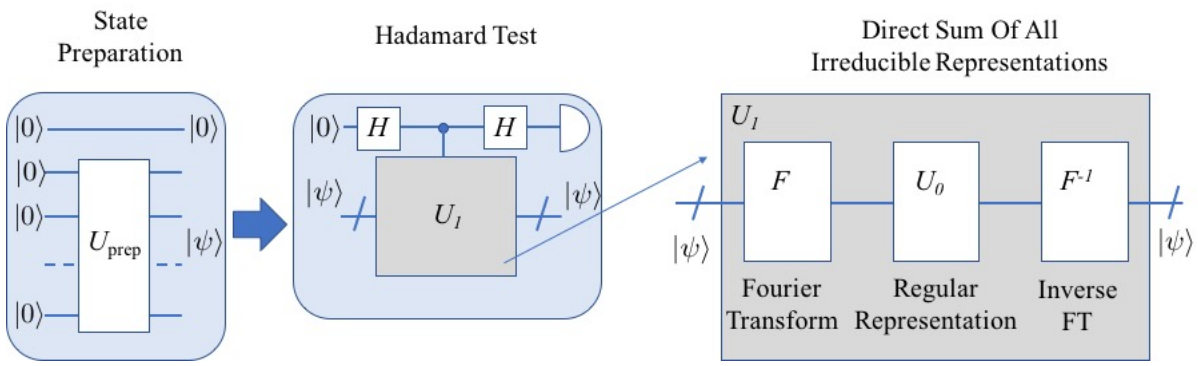


FIG. 14: Schematic diagram for the quantum algorithm

It is quite obvious: if quantum computer is capable to find expectation values of a unitary operator, it will solve the problem of finding the matrix elements of the regular representation of a group element. That will consist of, at least, two stages: the first stage is the state preparation, and the second is applying the unitary operator of the regular representation to that state. The unitary operator of the regular representation of an element of any group G_n can be created using a combination of only two type of operations: qubit flip ($|0\rangle \to |1\rangle$) and qubit swap ($|q_j q_i\rangle \to |q_i q_j\rangle$).

The regular representation is quite convenient: it is straightforward to find for any group, it carries all the information about the group, and a corresponding unitary operator seems to be easy to construct using standard quantum circuits. However, for groups with a large number of elements, it requires matrix multiplication between large matrices. Instead of regular representations one can use irreducible representations, which, literally, are matrix representations, which are impossible to equivalently reduce further. The direct sum of all irreducible representations (each has different dimensions d_ρ in general) is a block diagonal $n \times n$ matrix , where each irreducible representation appears exactly d_ρ times.

The Fourier transform pair over that group/representation can be introduced by decomposing each irreducible representation over the group elements and *vice versa*. Moreover, the direct sum of all irreducible representations can be decomposed as a regular representation conjugated by the direct and inverse Fourier transform operators [58].

B. Algorithm description

A quantum computer (gate-based) is a machine applying sequence of unitary operators to the qubit states, where each quantum gate is a unitary operator itself. The quantum algorithm calculating matrix elements $\langle\psi|\,\mathbf{U_1}\,|\psi\rangle$ of a unitary operator $\mathbf{U_1}$ is known as the Hadamard test, which is illustrated on Fig. 14. A unitary operator $\mathbf{U_1}$ can be represented as a 2^n-dimensional square unitary matrix, while the state $|\psi\rangle$ is a previously-prepared n-qubit state with respect to which matrix elements should be calculated. The algorithm requires one more control qubit, $|1\rangle$ or $|0\rangle$, in respect to which unitary operator $\mathbf{U_1}$ is either applied or not applied to the state $|\psi\rangle$ respectively. Such a mechanism is realized using the controlled-U_1 gate, which can be represented as a 2^{n+1}-square block-diagonal matrix, where the first block is a 2^n-identity matrix, and the second block is 2^n-square matrix of $\mathbf{U_1}$. The control qubit should be prepared as $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ or $\frac{|0\rangle-i|1\rangle}{\sqrt{2}}$ to calculate the real or imaginary parts of the matrix elements, respectively.

Let us demonstrate the direct calculation of the real part of the matrix element by direct implementing of gates on Fig. 14. Since in a real machine all qubits are initially in the ground state, first, we apply the Hadamard gate to the control qubit to create state $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$. At the same time we prepare state $|\psi\rangle$ from $|00..0\rangle$ using unitary operators of state preparation. Then, we have a quantum state onto which we apply the controlled-U_1 gate: $\frac{|0\rangle+|1\rangle}{\sqrt{2}}\,|\psi\rangle$, which gives: $\frac{|0\rangle|\psi\rangle+|1\rangle\mathbf{U_1}|\psi\rangle}{\sqrt{2}}$. By applying one more Hadamard gate to the control qubit we find our state: $\frac{|0\rangle(|\psi\rangle+\mathbf{U_1}|\psi\rangle)+|1\rangle(|\psi\rangle-\mathbf{U_1}|\psi\rangle)}{2}$. Hence, we will measure this state as $|0\rangle$ with probability: $P_0 = \frac{\langle\psi|+\langle\psi|\mathbf{U_1^+}}{2}\cdot\frac{|\psi\rangle+\mathbf{U_1}|\psi\rangle}{2} = \frac{1+Re\langle\psi|\mathbf{U_1}|\psi\rangle}{2}$. Hence, we find: $Re\,\langle\psi|\,\mathbf{U_1}\,|\psi\rangle = 2P_0 - 1$.

With the known Hadamard test algorithm, the problem for calculating matrix elements of an arbitrary unitary operator is reduced to the problem of effectively implementing it to the earlier-prepared state using quantum circuits. Hence, for the regular representation of any group U_0, where unitary operator is an n x n square matrix with only one non-zero element equal to 1 in each row, the solution is known for any group as a combination of $CNOT-$ and $Z-$

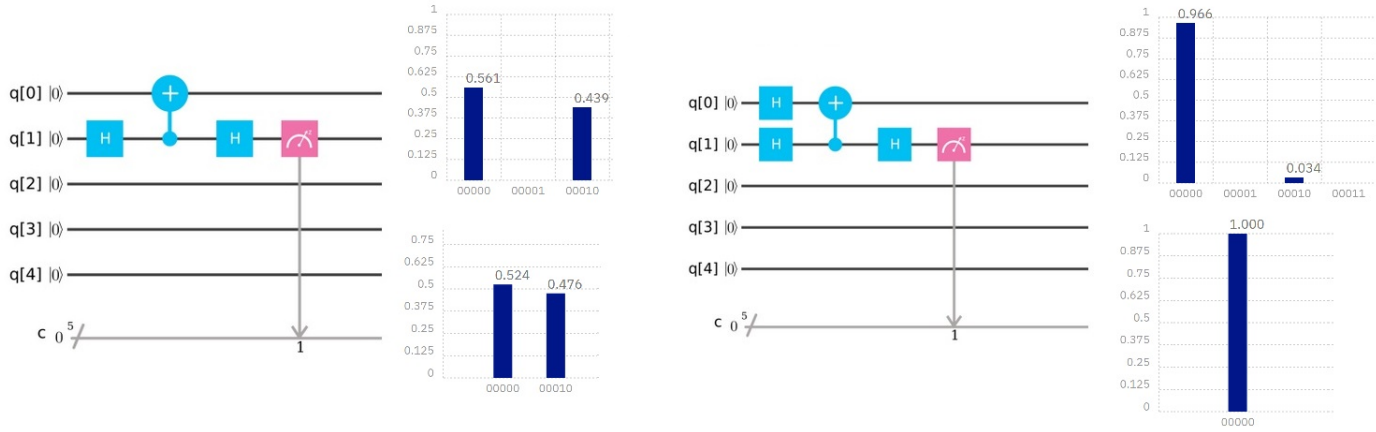


FIG. 15: Actual circuit implemented on IBM's 5-qubit computer (ibmqx4) for calculating matrix elements of the regular representation for the second element of the group S_2 and A_2 in respect to the state $|0\rangle$ on the left and $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ on the right. The expected probabilities to find a final state in the ground state are $(1+0)/2 = 0.5$ and $(1+1)/2 = 1$ respectively. The results of the 1024 runs on the actual chip (on the top) and the simulator (on the bottom) are presented on the right side of each circuit.

gates.

At the same time solutions for the direct sum of all irreducible representations U_1, which can be decomposed as consequent applications of the Fourier transform, the regular representation and inverse Fourier transform: $U_1(g) = F_1 U_0(g^{-1}) F_1^{-1}$, exists for any group where Fourier transform over that group can be effectively implemented using quantum circuits. Quantum circuits for the FOurier transform are already known for the symmetric group $S(n)$ [8], the alternating group A_n, and some Lie groups: $SU(n)$, $SO(n)$ [30], while solutions for other groups, hopefully, will be found in the future.

C. Algorithm implemented on IBM's 5-qubit computer

The actual gate sequence that we implemented on IBM's 5-qubit computer (ibmqx4) and IBM's quantum simulator to find matrix elements of the regular representation of the second element of the group A_2 and S_2 is shown in Fig. 15. The matrix for this representation is simply a NOT gate matrix. Hence, we have to use one $CNOT$ gate and two Hadamard gates, plus some gates to prepare state $|\psi\rangle$ from the ground state $|00\rangle$. We placed the the control qubit to the actual machine $|q_1\rangle$ qubit instead of $|q_0\rangle$, because of the specific machine architecture, where the first qubit can control the zero qubit but not *vice versa*. We could have used the original qubit sequence as reprinted on Fig. 14, by realizing $CNOT$ gate as a swapped $CNOT$ and four Hadamard gates, but it would add more gates to the circuit and potentially more computational errors rather than just a virtual swap of the qubits.

Calculating the irreducible representation for the same element of the group A_2 and S_2, which operator is represented by the same matrix as $Z-$gate, requires implementing controlled-Z gate, whose does not exist as an actual gate on the IBM quantum experience. However, it can be constructed using two Hadamard and one $CNOT$ gates as shown in Fig. 16. It is interesting to notice that the Hadamard gate is actually the Fourier transform operator over group S_2 and A_2, while the NOT-gate is a regular representation operator, as we mentioned earlier. Hence, such controlled-Z gate representation is in fact the decomposition of the irreducible representation to the regular representation using Fourier transform over that group.

D. Conclusion

We implemented a quantum algorithm for calculating matrix elements of two group representations: the regular representation and the direct sum of all irreducible representations for the second element of the symmetric group S_2. While this algorithm is quite simple for this particular case, it demonstrates capability of a real quantum computer to solve an actual problem today, and can be extended to solve the same problem for the symmetric group S_3 (6x6

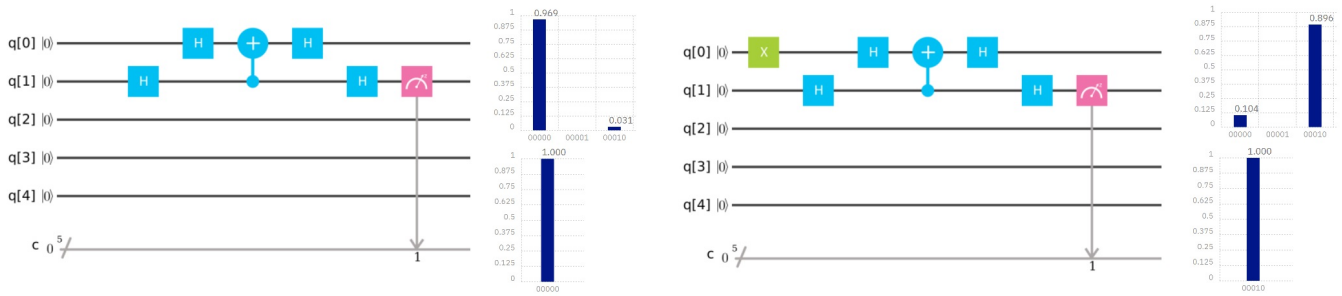


FIG. 16: Actual circuit implemented on IBM's 5-qubit computer (ibmqx4) for calculating matrix elements of the direct sum of the irreducible representations for the second element of the group S_2 and A_2 with respect to the state $|0\rangle$ on the left and $|1\rangle$ on the right. The expected probabilities to find a final state in the ground state are $(1+1)/2 = 1$ and $(1-1)/2 = 0$ respectively. The results of the 1024 runs on the actual chip (on the top) and the simulator (on the bottom) are presented on the right side of each circuit.

matrices) on the 5-qubit quantum computer (only 4 qubits are used) or even for the symmetric group S_7 (5040x5040 matrices) on the 16-qubit chip (14 qubits are used). However, an error correction routine will have to be implemented for such large circuits to ensure delivery of the correct results.

VII. QUANTUM VERIFICATION OF MATRIX PRODUCTS

A. Problem definition and background

Matrix multiplication is one of most important algorithm with applications such as solving eigenvalue problems and computing inverse of a given matrix. Therefore, the computational complexity of matrix multiplication is a subject of intense study. A faster algorithm for matrix multiplication implies a potential performance improvement for a variety of computational tasks. Strassen [100] first showed that two $n \times n$ matrices can be multiplied in time $n^{2+\alpha}$ ($\alpha < 1$). The best known bound to date is an algorithm with $\alpha \approx 0.376$ by Coppersmith and Winnograd [27], despite that, it remains an open problem to determine the true value of α. The so-called problem of matrix verification is defined as, verifying whether the product of two $n \times n$ matrices is equal to a third one. So far the best proposal can be done with high probability in time proportional to n^2 [41].

Ref. [3] was the first to study matrix verification for quantum computation. The authors use a quantum algorithm based on Grover's algorithm to verify whether two $n \times n$ matrices equal a third in time $O(n^{7/4})$, thereby improving the optimal classical bound of Ref. [41]. Ref. [17] presents a quantum algorithm that verifies a product of two $n \times n$ matrices over any integral domain with bounded error in worst-case time $O(n^{5/3})$ and expected time $O(n^{5/3}/min(w, \sqrt{n})^{1/3})$, where w is the number of wrong entries. This further improves the time performance $O(n^{7/4})$ from Ref. [3].

We briefly sketch the quantum algorithm in Ref. [3]. Let A, B, C be $n \times n$ matrices. First, partition the matrices B and C into $\sqrt{n}$ blocks of $\sqrt{n}$ columns each. It holds that $AB = C$ if and only if $AB_i = C_i$ for every i, where B_i and C_i are the sub-matrices of size $n \times \sqrt{n}$. The verification of $AB_i = C_i$ can be done with bounded error in time $On^{3/2}$ as follows: choose a random vector x of length $\sqrt{n}$, multiply both sides of the equation by x from the right side, compute classically $y = B_i x$ and $z = C_i x$, and verify the matrix-vector product $Ay = z$ by a Grover's search [48]. The search over n rows takes $O\sqrt{n}$ iterations and a verification of one row takes time n.

According to the above discussion, the key point in matrix verification is the Grover's search [48]. This quantum search algorithm is introduced in the following subsection.

B. Algorithm description

Consider an unsorted database with N entries. The algorithm requires an $N-$dimensional state space H, which can be supplied by $n = \log_2 N$ qubits. Consider the problem of determining the index of the database entry that satisfies some search criterion. Let f be the function that maps database entries to 0 or 1, where $f(x) = 1$ if and only if x satisfies the search criterion ($x = \omega$). We are provided with (quantum black box) access to a subroutine in the form of a unitary operator U_ω that acts as follows:

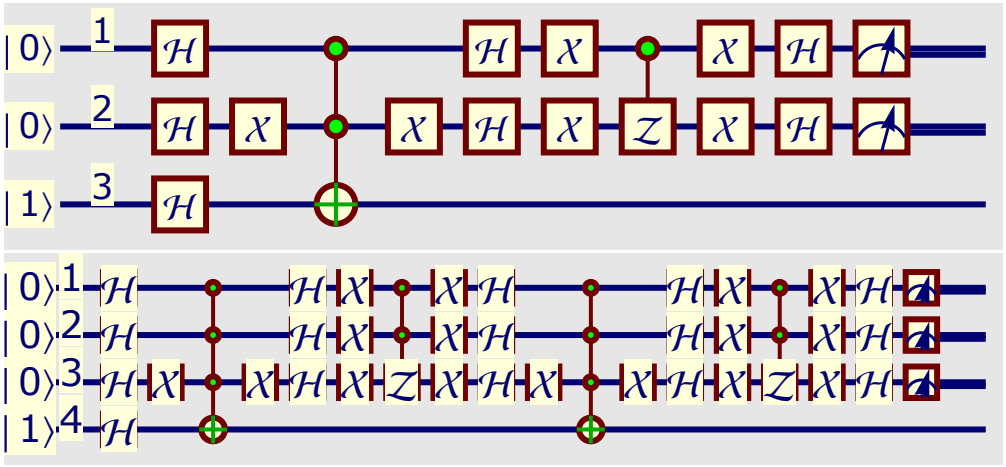


FIG. 17: Quantum circuit of Grover's search algorithm for (top) two qubits and (bottom) three qubits implemented on quantum computer. For two-qubit search, only 1 Grover iteration is need, while for three qubits case $n = 2$ Grover iterations are need. The actually circuit needs to decompose the control-control-NOT gate into several single or two-qubit gates.

Probability	Measure	State
1	$\{1, 0\}$	$\frac{-\|1,0,0\rangle+\|1,0,1\rangle}{\sqrt{2}}$

TABLE IV: Simulation results for two qubits search using Grover's algothrm. The quantum circuit is shown in Fig. 17.

$$\begin{aligned} U_\omega|x\rangle &= -|x\rangle, for\, x = \omega, iff(x) = 1; \\ U_\omega|x\rangle &= |x\rangle, for\, x \neq \omega, iff(x) = 0. \end{aligned}$$

The detailed algorithm steps are as follows. Let $|s\rangle$ denote the uniform superposition over all states

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

Then the operator

$$U_s = 2|s\rangle\langle s| - I$$

is known as the Grover diffusion operator.
Here is the algorithm:

1. Initialize the system to the state $|s\rangle$;
2. Perform the following "Grover iteration" $r(N)$ times. The function $r(N)$, which is asymptotically $O(N^{1/2})$, is described below.
   (a) Apply the operator U_ω.
   (b) Apply the operator U_s.
3. Perform the measurement.

The circuit shown in Fig. 17(top) gives the results for two-qubit search, where we search for $\{1, 0\}$).
The circuit in Fig. 17(bottom) is a three-qubit search for $\{1, 1, 0\}$.

Probability	Measure	State
$\frac{1}{128}$	$\{0,0,0\}$	$\frac{-\|0,0,0,0\rangle+\|0,0,0,1\rangle}{\sqrt{2}}$
$\frac{1}{128}$	$\{0,0,1\}$	$\frac{-\|0,0,1,0\rangle+\|0,0,1,1\rangle}{\sqrt{2}}$
$\frac{1}{128}$	$\{0,1,0\}$	$\frac{-\|0,1,0,0\rangle+\|0,1,0,1\rangle}{\sqrt{2}}$
$\frac{1}{128}$	$\{0,1,1\}$	$\frac{-\|0,1,1,0\rangle+\|0,1,1,1\rangle}{\sqrt{2}}$
$\frac{1}{128}$	$\{1,0,0\}$	$\frac{-\|1,0,0,0\rangle+\|1,0,0,1\rangle}{\sqrt{2}}$
$\frac{1}{128}$	$\{1,0,1\}$	$\frac{-\|1,0,1,0\rangle+\|1,0,1,0\rangle}{\sqrt{2}}$
$\frac{121}{128}$	$\{1,1,0\}$	$\frac{-\|1,1,0,0\rangle+\|1,1,0,1\rangle}{\sqrt{2}}$
$\frac{1}{128}$	$\{1,1,1\}$	$\frac{-\|1,1,1,0\rangle+\|1,1,1,1\rangle}{\sqrt{2}}$

TABLE V: Simulation results for three qubits search using Grover's algothrm. The quantum circuit is shown in Fig. 17.

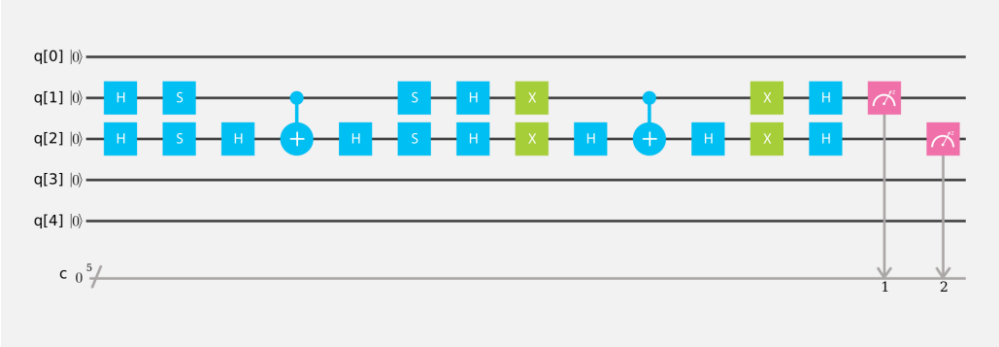


FIG. 18: Actual circuit implemented on IBM's 5-qubit simulator and quantum computer.

C. Algorithm implemented on IBM's 5-qubit computer

The actual gate sequence that we implemented on IBM's 5-qubit computer is shown in Fig. 18. This involved CNOT control-Z, and Hadamard gates. The decomposition of controlled-controlled-NOT into one- and two-qubit gates is done first by relating it to the phase gate S and Hadamard gate H. Our results are as follows. For the example, suppose that we are looking for configuration $\{1,0\}$, IBM's 5-qubit quantum computer with 1024 trials gave the results in Table VI.

Probability	Measure	State
0.024	$\{0,0\}$	$\|0,0\rangle$
0.943	$\{1,0\}$	$\|1,0\rangle$
0.016	$\{0,1\}$	$\|0,1\rangle$
0.017	$\{1,0\}$	$\|1,1\rangle$

TABLE VI: Results for two qubits search using Grover's algotrithm on IBM 5-qubits quantum computer. The quantum circuit is shown in Fig. 18.

Comparing the simulation results in Table IV and results from the IBM quantum computer in Table VI, we conclude that results from IBM quantum computer match our expectations very well.

VIII. GROUP ISOMORPHISM

A. Problem definition and background

The *group isomorphism* problem, originally identified by Max Dehn in 1911 [31], is a well-known decision problem in abstract algebra. Simply stated, it asks whether there exists an isomorphism between two finite groups, G and G'. Which, according to the standpoint of group theory, means that they are equivalent (and need not be distinguished).

To solve this problem using a quantum algorithm, we assume that each element can be uniquely identified by an arbitrary bit-string label. We also assume that a so-called group *oracle* can be used to return the product of multiple elements. That is, given an ordered list of group-element labels, the oracle will return the product label. In practice, this means that we must be able to construct a quantum circuit to implement $U_a : |y\rangle \to |ay\rangle$, for any $a \in G$. The use of an oracle to define an algorithm in this way is common in the discussion of quantum algorithms, and is sometimes referred to as a *black-box* setting [64].

In this section, we will focus our attention on the *abelian* group isomorphism problem, because it can be solved using a generalization of Shor's algorithm [94]. Here, abelian simply means that the operation ($\bullet$) used to define the group is commutative, such that $a \bullet b = b \bullet a$, for $a, b \in G$. Although Shor's approach is specifically intended to leverage a quantum order-finding algorithm to reduce the time-complexity of factoring, the procedure effectively solves the group isomorphism problem over cyclic groups. Using this relationship, Cheung and Mosca [21] have developed a theoretical quantum algorithm to solve the abelian group isomorphism problem by computing the decomposition of a given group into a direct product of cyclic subgroups.

B. Algorithm description

The procedure presented in algorithm 3 assumes the fundamental theorem of finite abelian groups, that they can be decomposed as a direct sum of cyclic subgroups of prime power order. This decomposition can then be used to test if an isomorphism exists between two groups.

Algorithm 1 Decompose($a_1, \ \ldots, a_k, q$), of Cheung and Mosca [21]

Input:
- A generating set $\{a_1, \ \ldots, a_k\}$ of G.
- The maximum order, q, of the generating set.

Output:
- The set of elements $g_1, \ \ldots, g_l$ from group G, with $l \leq k$.

Procedure:

Step 1. Define $g : \mathbb{Z}_q^k \to G$ by mapping $(x_1, \ \ldots, x_k) \to g(x) = a_1^{x_1} \cdots a_k^{x_k}$.
Find generators for the hidden subgroup K of $\mathbb{Z}_q^k$ as defined by function g.
Step 2. Compute a set $y_1, \ \ldots, y_l \in \mathbb{Z}_q^k / K$ of generators for $\mathbb{Z}_q^k / K$.
Step 3. Output the set $\{g(y_1), \ \ldots, g(y_l)\}$.

Since the procedure in Algorithm 3 is mostly classical, we shall treat the task highlighted in **Step 1** as the most critical for us to explore, because it calls for a solution to the hidden subgroup problem (HSP). This means that, given a function g that maps a finite group A onto a finite set X, we are asked to find a generating set for the subgroup K. For K to be the so-called *hidden subgroup* of A, we require that g is both constant and distinct on the cosets of K. On a quantum computer, this problem can be solved using a number of operations that is polynomial in $\log|A|$, in addition to one oracle evaluation of the unitary transform $U |a\rangle |h\rangle = |a\rangle |h \oplus g(a)\rangle$. The general procedure needed to complete **Step 1** of algorithm 3 is described in algorithm 2.

Algorithm 2 Solution to the hidden subgroup problem (for finite abelian groups). Based on Ref. [75]

Input:

- Two quantum registers.
- Elements of the finite abelian group A (or the generating set).
- A function g, such that $g : A \to X$, with $a \in A$ and $h \in X$.

Output:

- The generating set for the hidden subgroup K.

Procedure:

Step 1. Create initial state.

Step 2. Create superposition between resisters.

Step 3. Apply unitary operation (U) for function $g(a)$.

$$\to \frac{1}{\sqrt{|A|}} \sum_{a \in A} |a\rangle \, |g(a)\rangle \tag{35}$$

Step 4. Apply inverse Fourier transform.

$$\to \frac{1}{\sqrt{|A|}} \sum_{l=0}^{|A|-1} e^{2\pi i l a/|A|} \, |\hat{g}(l)\rangle \tag{36}$$

Step 5. Measure the phase from first register.

$$\to l/|A| \tag{37}$$

Step 6. Sample K from l / | A |.

Like the period-finding approach used in quantum factorization, Algorithm 2 is heavily based on the concept of phase estimation. Note that the Fourier transform in Eq. 36 represents $a \in A$ indexed by l. The key concept of the procedure is that $|\hat{g}(l)\rangle$ has nearly zero amplitude for all values of l, except those which satisfy

$$|K| = \sum_{h \in K} e^{-2\pi i l h/|A|}, \tag{38}$$

and that knowledge of l can be used to determine both the elements and generating set of K. As discussed by Nielsen and Chuang [76], the final step in algorithm (2) can be accomplished by expressing the phase as

$$\to e^{2\pi i l a/|A|} = \prod_{i=1}^{M} e^{2\pi i l_i' a_i/p_i}. \tag{39}$$

for $a_i \in \mathbb{Z}_{p_i}$, where p_i are primes, and $\mathbb{Z}_{p_i}$ is the group containing integers $\{0, 1, \ \ldots, p_i - 1\}$ with the operator being addition modulo p_i.

The quantum circuit needed to solve the HSP is schematically illustrated in Fig. 19. This simplified circuit includes steps 1-5 of algorithm 2, and makes it clear that all forms of the HSP (order-finding, period-finding, discrete logarithm, etc.) are extensions of quantum phase estimation.

C. Algorithm implemented using QISKit

Since the generalized group isomorphism problem is somewhat complex, we will focus here on the implementation of the HSP circuit fragment illustrated in Fig. 19. We also chose a specific instance of the HSP: the problem of finding the period of $a\%n$. In Fig. 20, the basic outline of the code needed for this specific problem is illustrated using the python-based QISKit interface.

Like most instances of the HSP, one of the most challenging practical tasks of finding the period of $a\%n$ on a quantum computer is the implementation of the oracle. The details of the oracle are not explicitly shown in the QISKit snippet, but for the required $Ca\%15$ operations, one can simply used the circuits developed by Markov and Saeedi [88]. The code in Fig. 20 also assumes that a function $qft\_inv()$ will return the gates for an inverse quantum Fourier transform, and that a classical *continued fractions* algorithm can be used to convert the end result (a phase) to the desired integer period.

Although the specific procedure outlined in Fig. 20 can be directly implemented using the IBM QISKit interface, the resulting QASM code will not lead to impressive results on the IBMX4 (or IBMX5). This is because the generated

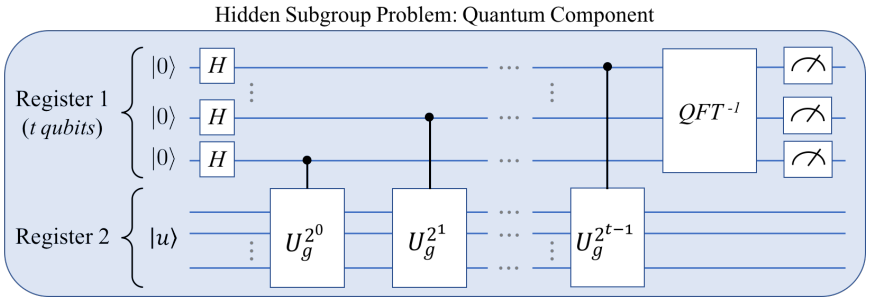


FIG. 19: Basic phase-estimation quantum circuit needed to solve the general hidden subgroup problem in algorithm 3. Here, $|u\rangle$ is an eigenstate of the unitary oporator U.

```
#=================================================================#
#----------  Finding period (r) of a % N, with N=15  -------------#
#=================================================================#
def findperiod(a, N=15, nqubits1, nqubits2):

    # Create QuantumProgram object, and define registers and circuit
    Q_program = QuantumProgram()
    qr1 = Q_program.create_quantum_register("qr1", nqubits1)
    qr2 = Q_program.create_quantum_register("qr2", nqubits2)
    cr1 = Q_program.create_classical_register("cr1", nqubits1)
    cmod15 = Q_program.create_circuit("cmod15", [qr1, qr2], [cr1])

    # Apply a hadamard to each qubit in register 1
    # and prepare state |1> in regsiter 2
    for j in range(nqubits1): cmod15.h(qr1[j])
    cmod15.x(qr2[nqubits2-1])

    # Loop over qubits in register 1
    for p in range(nqubits1):

        # Calculate next 'b' in the Ub to apply
        # ( Note: b = a^(2^p) % N ).
        # Then apply Ub
        b = pow(a,pow(2,p),N)
        CxModM(cmod15, qr1, qr2, p, b, N, nqubits1, nqubits2)

    # Perform inverse QFT on first register
    qft_inv(cmod15, qr1, nqubits1)

    # Measure each qubit, storing the result in the classical register
    for i in range(n_qr1): cmod15.measure(qr1[i], cr1[i])
```

FIG. 20: Simple implementation of the quantum period-finding algorithm using the QISKit interface to the IBM Quantum Experience.

circuit is long-enough for decoherence error and noise to ultimately dominate the measured state. In other words, the physical hardware requires further optimization to reduce the number of gates used between the initial state preparation and the final measurement.

D. Conclusion

In conclusion, the group isomorphism problem can be solved for finite abelian groups by computing the decomposition of a given group into a direct product of cyclic subgroups. This approach, in turn requires the solution of the hidden subgroup problem, for which efficient quantum algorithms are already known. In this section we have discussed the details of a quantum algorithm for the hidden subgroup problem, and demonstrated that a specific instance of the algorithm (that of period finding) can be implemented using the IBM QISKit interface. For the period-finding algorithm to be successful on the real physical hardware, however, further circuit optimizations are needed.

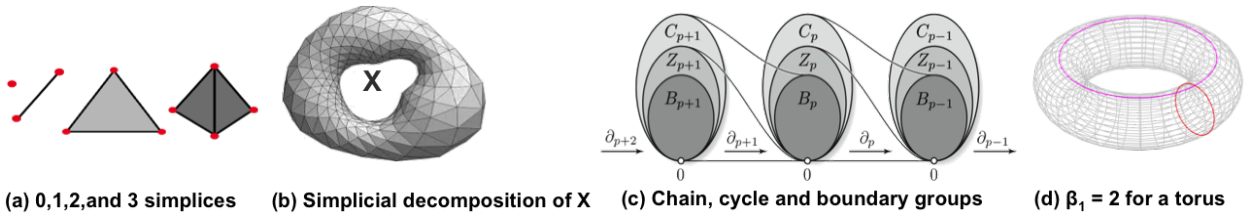


FIG. 21: Examples of simplices, simplicial decomposition of a topological space X, relationships among groups C_p (chains), Z_p (cycles), and B_p (boundaries) under the action of boundary homomorphisms ∂_p, and the example of the torus.

IX. QUANTUM PERSISTENT HOMOLOGY

A. Problem definition and background

Big data analysis often involves large numbers of multidimensional data points. Understanding their structure can lead to insights into the processes that generated them. Data clustering is closely related to spatially connected components. Other features such as holes and voids and their higher dimensional analogs that characterize the distributions of data points are useful for understanding their structure. Persistent homology connects data points across scales to reveal the most enduring features of datasets. Methods from algebraic topology are employed to build simplicial complexes from data points, and the topological features of these simplicial complexes are extracted by linear algebraic techniques. However, such an investigation on a set of n points leads to storage and computational costs of $O(2^n)$ as there is a combinatorial explosion in the number of simplices generated by n points. Thus representational and computational efficiency has to be greatly enhanced for viability. Quantum algorithms provide such efficiency by superposing 2^n simplex states with only n qubits and implementing quantum parallel computations. The study of such a quantum algorithm, proposed by Lloyd et. al [67] is the focus of this report.

Data points $P = \{p_0, \ldots, p_{n-1}\}$ can be envisaged as vertices of a *simplicial decomposition* of a subset X. An oriented $k-$simplex $\sigma_k = [p_{j_0}, \ldots, p_{j_k}], 0 \leq j_0 < j_1, \ldots, < j_k \leq n-1$, is the convex hull of $k+1$ points, and the simplicial complex is comprised of all the simplices. Thus, a 0-simplex is a vertex, a 1-simplex is an edge, a 2-simplex is a triangle, a 3-simplex is a tetrahedron, and so on. Determining which distance scales ϵ of vertex connectivity capture enduring topological features is the goal of Persistent Homology [67]. The numbers of various topological features at any scale are obtained from algebraic structures involving the simplices.

Define the kth chain group C_k as the set of all formal integer linear combinations of k-simplices: $C_k = \{\sum_i a_i \sigma_k^i | a_i \in \mathbb{Z}\}$. C_k is an abelian group generated by the k-simplices. Further, define boundary operators $\partial_k : C_k \rightarrow C_{k-1}$ between chain groups as group homomorphisms whose action on a k-simplex $\sigma_k = [p_{j_0}, \ldots, p_{j_k}]$ is given by $\partial_k \sigma_k = \sum_{i=0}^{k} (-1)^i [p_{j_0}, \ldots, p_{j_{i-1}}, p_{j_{i+1}}, \ldots, p_{j_k}]$ (i.e., the ith vertex is omitted from σ_k, $0 \leq i \leq k$, to get the $k+1$ oriented ith boundary $(k-1)$-simplex faces). With this, every k-chain $c_k^i \in C_k$ gives rise to a $(k-1)$-chain $c_{k-1}^i \in C_{k-1}$. A chain $c \in C_k$ such that $\partial_k c = 0$, where 0 is the null chain, is called a k-cycle. Also, $\partial_k \partial_{k+1} \equiv 0$. That is to say, the boundary of a boundary is the null chain 0, since the boundary of every $k+1$-chain is a k-cycle. $Z_k = Ker(\partial_k)$ is the subgroup of C_k consisting of all k-cycles, and $B_k = Image(\partial_{k+1})$ is the subgroup of C_k consisting of boundaries of all $(k+1)$-chains in C_{k+1}. Clearly, $B_k \subsetneqq Z_k$. The relationships between chain groups, cycles and boundaries as established by the boundary homomorphisms is illustrated in Fig.21(c). The kth *Betti number* β_k of a topological space X is defined as the number of linearly independent k-cycles that are not boundaries of $(k+1)$-chains, and characterizes the topological features at dimension k. For instance, β_0 is the number of connected components of X, β_1 is the number of 1-dimensional holes, β_2 is the number of voids, and so on. The kth *Homology Group* of X is defined as the quotient group $H_k(X) = Z_k(X)/B_k(X)$, whereby β_k is the number of generators of $H_k(X)$.

Equivalently, the kth Betti number β_k is the dimension of the kernel of the combinatorial Laplacian operator $\Delta_k = \partial_k^\dagger \partial_k + \partial_{k+1} \partial_{k+1}^\dagger$. $\beta_k = dim(Ker(\Delta_k))$. This allows the computation of Betti numbers by finding the null space of a linear transformation. Lloyd's quantum algorithm [67] diagonalizes the Laplacian to compute Betti numbers.

B. Quantum Algorithm Description

A quantum algorithm for calculating Betti Numbers was presented in [67]. The algorithm: (1) Applies Grover's Algorithm with scaled distances between points as input. (2) Computes an outer product from the output of the

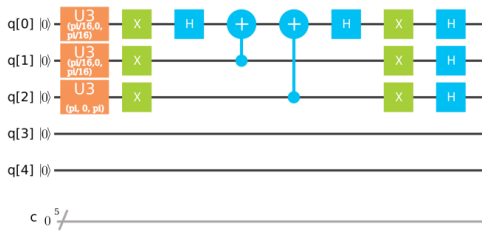


FIG. 22: Grover's Algorithm circuit implemented on the 5 qubit quantum computer showing 3 qubits being used with the multiple solution version of Grover's Algorithm. U3 gates are used to input the scaled distances between points. There were no details provided for scaling or state preparation.

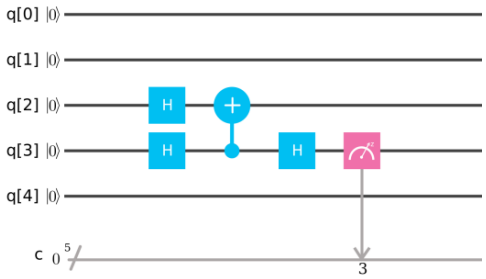


FIG. 23: Quantum Phase Estimation Algorithm circuit implemented on the 5 qubit quantum computer showing how the quantum density matrix implemented on qubits 0, 1, and 2 would be applied to obtain a classical measurement on qubit 3.

quantum states calculated with Grover's algorithm to produce a quantum density matrix ρ, and (3) performs Quantum Phase Estimation (+ form) to obtain the probabilities of an eigenvalue to be multiplied with the number of simplices to obtain input for the calculation of a Betti Number. Working with only 5 qubits, implies that the largest number of points n that could be processed at once is constrained by $n(n-1)/2 \leq 5$. Thus only 3 points at a time could be processed on the 5 qubit quantum computer. The implementation of Grover's Algorithm is shown in Fig. 22, only one iteration of Grover's algorithm is shown, more iteration forms were produced however there was no discussion of how many iterations of Grover's Algorithm would need to be performed prior to calculating the quantum density matrix. The output of this part of the algorithm is a quantum distribution of simplices.

Calculating the quantum density matrix could not be accomplished by the IBM machine due to the lack of Quantum RAM needed for the algorithm. Options considered to circumvent this problem included implementing a quantum algorithm for computing the outer product to form an 8x8 quantum density matrix. This was abandoned as the sheer size of quantum algorithms to implement four qubit addition [107] was well beyond the 5 qubits available on the quantum computer. Message boards' suggestion to perform Grover's algorithm 64 times and reassemble the output into a density matrix was also not viable due to decoherence. Had a quantum density matrix been produced, the Quantum Phase Estimation (+ form) would have been applied to find the probabilities of eigenvalues of the boundary operator. This would then be multiplied times the number of simplices and used as input to find the Betti Number. The Quantum Phase Estimation Algorithm's quantum circuit is shown in Fig. 23.

In order to check the coherence of the quantum 5 qubit computer a study was designed. All five qubits were flipped in the first timestep and measurements were then taken place approximately every five timesteps throughout the 74 available timesteps in the quantum composer. The quantum algorithm applied is shown in Fig. 24 showing the use of the Idle gates for 5 timesteps after the qubits are flipped with the X gate.

The data was collected for all the timesteps, processed into a form to evaluate the coherence percentages for all individual qubits and for all qubits combined. The results are depicted in Fig. 25 showing the decoherence rates with quantum composer timesteps. Note that although the coherence rates are fairly high for individual qubits, the overall qubit coherence percentages are far less. This is because a single qubit decohering also decoheres the entire 5 qubit quantum computer. This should be considered to determine how many qubits will actually be usable if a machine is purchased. It is also interesting that qubit 3 decoheres at a faster rate than the other qubits past timestep 15.

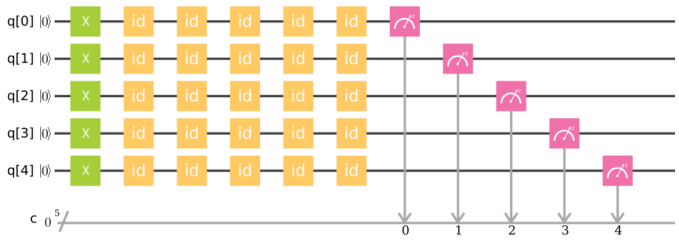
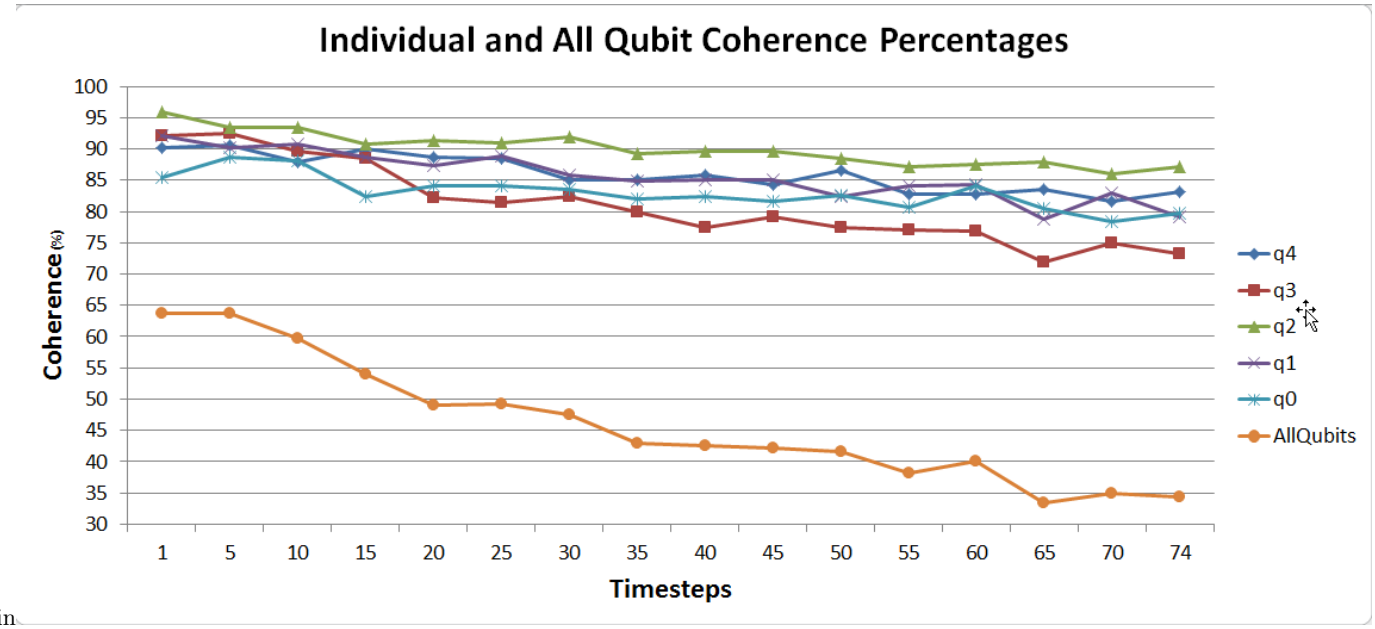


FIG. 24: The quantum algorithm applied to estimate the Decoherence time of the 5 qubit quantum computer as implemented in the Quantum Composer to measure the decoherence after 5 timesteps. Note that this produces an optimistic estimate as the "id" quantum gates utilize minimal time whereas other gates, such as CNOT and U3 gates, could use more time.



inin

FIG. 25: Timesteps available with the Quantum Composer are shown on the x axis, ranging up to 74. Coherence percentage is calculated for all individual qubits and is plotted on the y axis. Coherence percentages of all 5 qubits combined is also shown on the y axis. Note that the coherence percentage rate falls below 50 percent between the 15th and the 20th timesteps.

C. Conclusion

In conclusion, we attempted to implement a quantum algorithm for performing persistent homology on IBM's 5-qubit computer. Absent a Quantum RAM, the algorithm could not be implemented in full. However, two of the three steps described were completed. Concern over decoherence time on the 5 qubit machine motivated us to study it. This study showed that although the individual percentage of coherence all remained above 70 percent throughout the 74 timesteps available, coherence dropped below 50 percent before the 20th timestep was reached. This is an optimistic study by design and so the actual decoherence time is likely to be less with the implementation of common quantum algorithms. Finally, this exercise showed the need for Quantum RAM for a Universal Quantum computer to be useful. The absence of a loop construct on a quantum computer is severely limiting, and quantum algorithms that would be simple in classical computing are very difficult and often need many more qubits than initially expected.

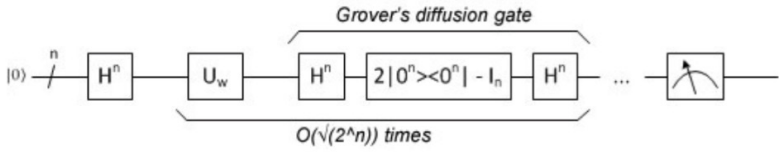


FIG. 26: Schematic diagram for the Grover's search quantum algorithm (picture from [36]). The first step is to initialize all qubits using the Hadamard gate on each for a uniform superposition of all states. This is followed by running the Grover iteration $\sqrt{(N)}$ times, where $N = 2^n$ is the number of items in the search. The Grover iteration consists of running the oracle, U_w, followed by a diffusion operator. The oracle is a unitary matrix gate that encodes the unstructured list items. The diffusion operator performs inversion about the mean. The last step is measurement of the qubit results.

X. QUANTUM ALGORITHMS FOR GRAPH PROPERTIES

A. Problem definition and background

Quantum computing is an intersection of mathematics, computer science, and physics. As a computer scientist, gaining an understanding the quantum computing gate model approach involved reading books [65, 75, 86, 109], tutorials [53] and relevant publications, experimenting with simulators and qubit hardware [52], and developing quantum programs [20].

This learning experience focused on quantum algorithms for graph properties in the adjacency matrix model starting from the Quantum Zoo [57]. Quantum algorithms for graph properties using the adjacency matrix (as part of an oracle) have been published for minimum spanning tree, shortest path, deciding if a graph is bipartite, detecting cycles, finding subgraphs (such as a triangle), maximal clique, and many more. Each typically involves the use of Grover's search [48] with an oracle constructed from the adjacency matrix. They are measured in quantum query complexity, sometimes providing polynomial speedup over classical algorithms.

Grover's algorithm is also used to solve many math problems [84]. In particular, algorithms of interest include the triangle problem [70], finding cycles [23], and finding maximal cliques [108].

Additionally, quantum random walk algorithms can also be used to search and find graph properties [24, 34, 38, 59, 60, 69]. These are similar in principle to classical random walks. Quantum random walk algorithms come in two forms, discrete time quantum walk and a continuous time quantum walk [59]. The discrete form operates in a step-wise fashion, requiring multiple copies of a set of gates per step. The continuous form uses a transition matrix that is processed into a Hamiltonian matrix that can be solved by quantum amplitude amplification. Quantum random walks are used to walk a graph or cycle [34, 60], search for marked verices [38], and s-t connectivity [60].

B. Quantum Search using Grover's Algorithm

Grover's search is like finding a needle in a haystack. Background and explanations for Grover's algorithm are found in [36, 45, 54]. The flow is shown in Fig. 26. The oracle can be different depending on the problem being solved, while the diffusion operator remains the same.

Grover's algorithm consists of the following steps.

1. Start with state $|0\rangle$.
2. Apply Hadamard on all n qubits.
3. Repeat $\sqrt{2^n/t}$ times, where t is the number of true answers.
   (a) Apply the oracle.
   (b) Apply the diffusion operator.
4. Measure the qubits.

Quantum amplitude amplification is a generalization of the Grover's search algorithm that can fit into many different algorithmic contexts, such as optimization.

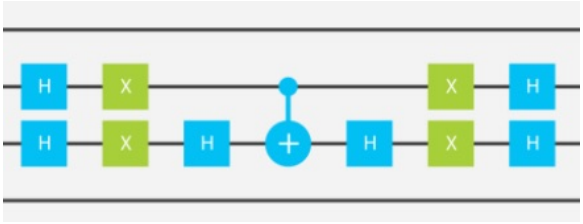


FIG. 27: The Grover diffusion operator is built out of Hadamard gates surrounding an operation that inverts the phase of the first state. This is shown for a two qubit system.

a. The Oracle. The oracle as part of Grover's search doesn't search through lists, it evaluates items by their indices. Grover's algorithm takes a function, searches through the implicit list, and (with high probability) returns the single index that causes the function to return true [45].

The internals of the oracle are typically considered a black box. The oracle matrix can be generated classically ahead of time or the oracle can have an unstructured list or database to back it up. Even quantum memory has been suggested [16]. More challenging, an oracle can be constructed as a circuit representing a predicate or function [103]. In this case creating the oracle may take longer than running the search.

A graph properties oracle can be assembled classically prior to running Grover's search or as a circuit based on the adjacency matrix and linear algebra transformations. For example, a quantum circuit for finding maximal cliques in a network with n=3 nodes, requires an oracle workspace of n^2 data qubits and n^2 ancilla qubits (see [108]). Each oracle call requires execution of $6n^2$ Toffoli gates and $2n$ CNOT gates. Testing a query such as $|110>$, which is $\{1, 2\}$, an intersection of columns 1 and 2 of $A + I$, where A is the adjacency matrix, yields a common neighbor set of $\{1, 2\}$. This is compared against the query which flips the sign of the state relative to all other states if true. An oracle such as this can be run on a simulator, but requires too many qubits to run on actual qubit hardware.

Quantum algorithms for finding a triangle, quadrilateral, longer cycles, and arbitrary subgraphs [23] typically use the adjacency matrix A to create the oracles classically, but circuits could also be constructed.

b. The Diffusion Operator. The soul of Grover's algorithm is the diffusion operator that computes the average of all the amplitudes, and then inverts all of the amplitudes through that average (and additionally negates all the amplitudes). The circuit is shown in Fig. 27. This is also known as inversion about the mean or inversion above the average. This boosts the separation of the phases causing the amplitude of the desired state to be separated from the other states.

C. Quantum Random Walk

Quantum random walk algorithms are an analogue to the classical random walk and Markov chain. They can be used to search unsorted data as does Grover's search, walking cycles, s-t connectivity, and more. The discrete time quantum walk algorithm requires the use of one or more coin qubits representing the number of valency or number of choices to move to from each graph node. This represents a random coin toss to decide the choice from a vertex. Only one coin qubit is required for walking a line of vertices or a cycle. Each node is represented by a binary string, where the next step choice has a one bit difference (Hamming distance).

Quantum random walk consists of the following steps (for walking around a 4-node graph 29):

1. Start with state $|0\rangle$ on starting 2 vertex qubits.
2. Coin operator is a Hadamard (or ciruit) on coin qubit (coin toss).
3. Repeat p times, where p is the number of hops in the path.
   (a) Apply shift operator to the first vertex qubit based on coin qubit.
   (b) Apply X or shift gate to the coin qubit for a bit flip.
   (c) Apply shift operator to the second vertex qubit based on coin qubit.
4. Measure the vertex qubits.

This can be modified for graphs of different sizes. The coin operator can be a Hadamard gate or a sub-circuit that results in a random bit generator. The shift operator can be simple as described above or can be a more complicated circuit that selects the next vertex in the path based on the representation used.

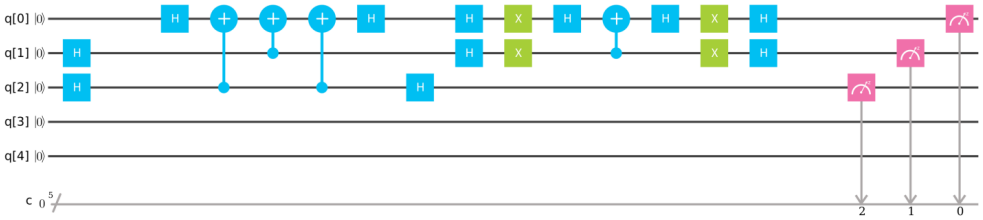


FIG. 28: This Grover Search circuit initializes the qubits, queries an oracle using an extra ancilla qubit, runs the diffusion operator, and produces 2 possible marked results.

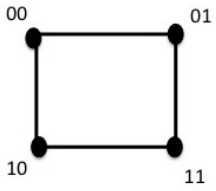


FIG. 29: A graph of 4 nodes in the form of a square is used for the random walk algorithm. The starting vertex is labeled 00. The next possible vertex choices vary by 1-bit in their labels, 01 and 10. The next possible vertex choices are 00 and 11. The quantum walk algorithm will walk around the graph.

D. Algorithm Implementations on IBM's Quantum Experience

Simple examples for Grover's Search and Quantum Walk were implemented and run on the simulator and 5-qubit quantum computer. Problem description, circuit, and results are shown.

The Grover Search example explores the oracle as a circuit and the use of an extra ancilla qubit as part of the oracle workspace. The diffusion circuit remains the same as described above in Fig. 27. In this example there are 2 marked results. There are 4 items in an unstructured list representing a graph property. The oracle uses 1 ancilla qubit that could be part of an adjacency matrix. Qubit q[0] is initialized to $|0\rangle$. A Hadamard gate is applied to q[1] and q[2]. A CNOT with the ancilla q[2] as a control may flip q[0]. The oracle circuit is reversed and then the diffusion operator is applied to q[0] and q[1]. The circuit is shown in Fig. 28. This circuit results in 01 and 11 similarly for the simulator and the 5-qubit chip.

The Quantum Walk example walks around a 4-npde graph in the form of a square (see Fig. 29). Starting at vertex 00 in 30, step 1 chooses 01 or 10 with similar probability. Step 2 chooses 11. Step 3 chooses 01 or 10. And step 4 brings us back to 00. Results of steps 1-3 were obtained from shorter circuits. The results are the same on the IBM quantum simulator and 5-qubit chip, ibmqx4.

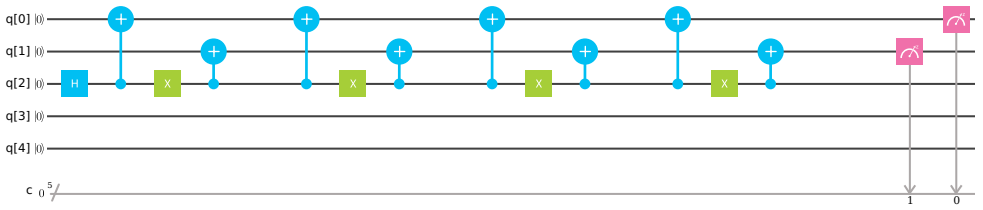


FIG. 30: Actual quantum walk circuit implemented on IBM's 5-qubit simulator and quantum computer. The coin operator on qubit q[2] is a Hadamard. The 4 steps to walk the graph in Fig. 29 are shown. Each step consists of a shift operator (CNOT) on qubit q[0], followed by a bit flip (X) on the coin qubit q[2], and a shift operator (CNOT) on the second qubit q[1]. Only one of the 2 quibits (q[0] or q[1]) is flipped in any step.

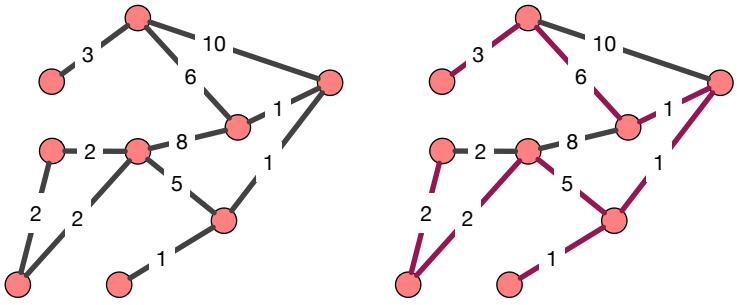
E. Conclusion

Quantum algorithms for graph properties using the adjacency matrix were explored based on Grover Search and Quantum Walk. Due to the limited number of qubits available and the many basic gates required when decomposing 3-qubit, 4-qubit, etc. gates (such as Controlled-Z and Toffoli), only simple examples are possible that can run on quantum hardware. Some larger cicuits can be built to run on the simulator. Future plans include exploring graph properties quantum algorithms similar to the finding of maximal cliques [108].

XI. QUANTUM MINIMAL SPANNING TREE

A. Problem definition and background

A common problem in network design is to find a minimum spanning tree. Suppose we are responsible for maintaining a simple network of roads. Unfortunately, each segment needs repair and our budget is limited. What combination of repairs will guarantee the network remains connected? Figure 31 shows a model of a simple road network as a graph, together with a minimal spanning tree.



(a) The weighted graph model. (b) A minimal spanning tree.

FIG. 31: A graph modeling repair costs of a simple transportation network (a) together with (b) its minimal spanning tree (the red edges). The sum of the weights of the edges in the minimal spanning tree is 21.

Formally, a graph $G = (V, E)$ consists of a set V (the nodes) and a set E consisting of pairs of nodes. A graph is connected if between any two nodes there exists a path. A spanning tree of a connected graph $G = (V, E)$ is the graph $T = (V, E_T)$ where $E_T \subset E$ and T contains no cycles (i.e., there is exactly one path between any two vertices). It is not had to see that a graph T is a spanning tree if and only if T is connected and has n nodes and $n - 1$ edges. A weighted graph is a graph $G = (V, E, w)$ where w is a map on the edges $w : E \to \mathbb{R}$. A minimal spanning tree of a graph G is then a spanning tree $T = (V, E_T)$ which minimizes

$$\sum_{e \in E_T} w(e). \tag{40}$$

B. Algorithm description

Algorithmically, a graph is usually presented in one of two ways: either as a list of edges or as an adjacency matrix. We consider the case where G is presented as a list of edges. A quantum algorithm for finding a minimal spanning tree of an input graph is given in [35]. This algorithm requires only $O(\sqrt{nm})$ queries where n is the number of nodes and m the number of edges in the graph. Classically, the best algorithms run in time $O(m \log n)$. This is the time complexity of Borůvka's algorithm [13]. The quantum algorithm combines Borůvka's algorithm together with the quantum search algorithm of Grover [48].

Borůvka's algorithm builds a collection of disjoint trees (i.e., a forest) and successively merges by adding minimal weight edges. The first two steps of the algorithm are shown in Figure 32. Formally we have

1. Let $\mathcal{T}$ be the collection of n disjoint trees, each consisting of exactly one node from the graph G.

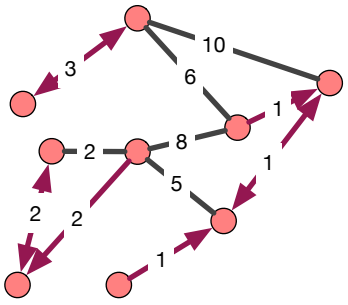


FIG. 32: The first two steps of Borůvka's algorithm. Starting with each node as a distinct tree, find the minimal weighed edge between each tree and the rest of the trees. The direction of the red edges indicates the edge is the minimal weighted edge for the source node. The components connected by red edges (disregarding the directions) will form the trees at the start of the second run of step (2).

2. Repeat:

   (a) For each tree T_i in $\mathcal{T}$ find the minimal weighted edge, e_i, of G between T_i and the other trees of $\mathcal{T}$.

   (b) Merge the trees $\{T_i \cup \{e_i\}\}$ so that they are disjoint: set this new collection to $\mathcal{T}$.

If there are k trees in $\mathcal{T}$ at a given iteration of Step (2), then the algorithm performs k searches for the respective minimal weighted edges. As the trees are disjoint, we can perform the k searches in one sweep by inspecting each of the m edges of G once. As there will be at most $\log n$ iterations of Step (2), this results in a running time of $O(m \log n)$. The quantum algorithm takes advantage of the Grover search algorithm, to speed up the searches in Step (2).

Grover's algorithm describes a general method to find a given object within an unstructured list of N objects with only $O(\sqrt{N})$ queries to the list. Grover also described how to generalize this method to searching a list for any of k objects with only $O(\sqrt{N/k})$ queries to the list. The key ingredient to implementing Grover's algorithm is the so-called oracle which performs the queries to the list. We call those elements for which we are searching the "marked" elements of the list. Formally, suppose $N = 2^n$ and let $f : [N] \to \{0, 1\}$ be defined on the indices of the list to be searched such that

$$f(i) = \begin{cases} 1 & \text{if the } i^{\text{th}} \text{ element is marked} \\ 0 & \text{otherwise} \end{cases} \tag{41}$$

The oracle U_f acts on $n+1$ qubits: the first n qubits describe a superposition of the indices to search; the last qubit is ancillary.

$$U_f \left(|x\rangle \frac{|0\rangle - |1\rangle}{2} \right) = (-1)^{f(x)} |x\rangle \frac{|0\rangle - |1\rangle}{2} \ . \tag{42}$$

While Equation 42 is often used in the literature (see for example the excellent book [75]), there is a more concise unitary oracle which acts only on n qubits:

$$U_f \left(|x\rangle \right) = (-1)^{f(x)} |x\rangle \ . \tag{43}$$

The advantages of this formulation are two-fold: on the small quantum computer's available each qubit counts, and the matrix form of Equation 43 is particularly easy to visualize as it is a diagonal matrix with ones and negative ones.

Of course, hard-coding U_f defeats the purpose of a fast search. Ideally one would implement a f as a circuit which evaluated to 1 after a short computation. In the algorithm above, we need to find the minimal element of an appropriate list. Clearly this can not be implemented directly as f without actually inspecting each of the N list elements. Luckily, there is a simple work around given by Durr et al [35] which involves multiple calls to the Grover algorithm as follows.

1. Initialize j to a random element of the list.

2. Repeat

(a) Search for $i \leq j$ in the list.

(b) Update $j := i$.

A simple probabilistic analysis shows that altogether only $O(\sqrt{N})$ queries are called in expectation to find the minimal element [35].

C. Algorithm implementation on IBM's 5-qubit computer

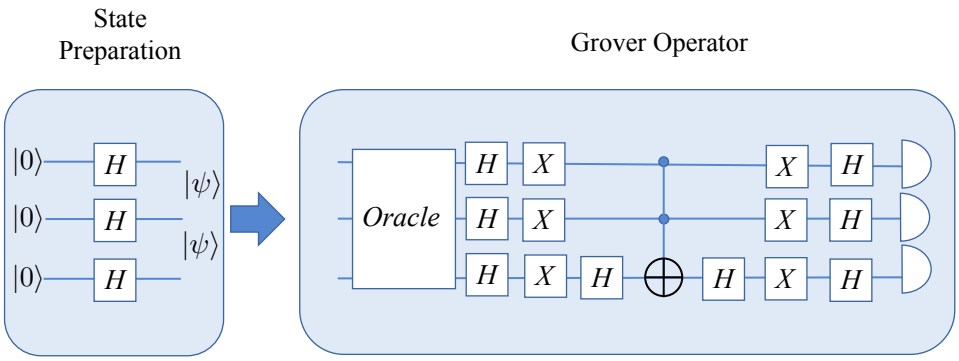
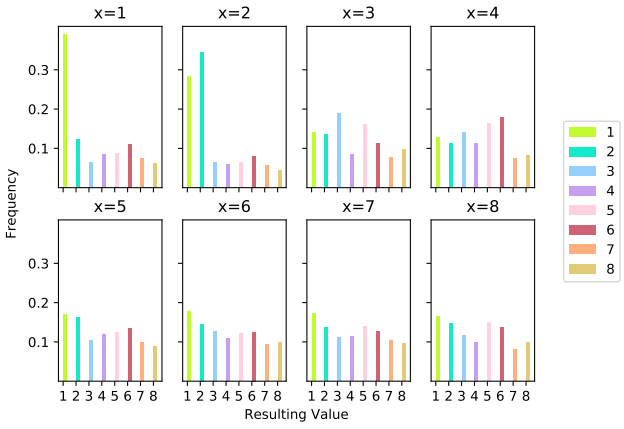
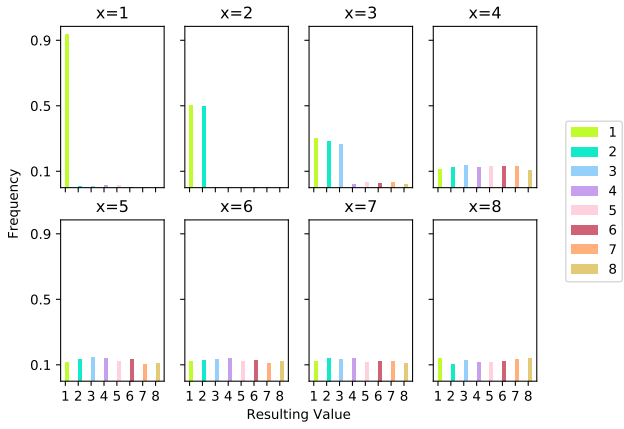


FIG. 33: A schematic of Grover's Algorithm

We did not implement the full algorithm due to space constraints on the IBM computer. Even to successfully implement a minima finding algorithm, at least 6 qubits would be necessary to compare 2 3-bit numbers. Therefore we implemented the minima finding algorithm by hard coding the unitary operator U_f for each of eight possible functions f: $\{f(i) = 1 \text{ if } i \leq x : 1 \leq x \leq 8\}$. The results are shown in Figure 34. The QASM code for implementing $f(i) = 1$ if $i \leq 2$ required just under 100 lines of code (more than 100 individual gates.) The results, even when using the simulator are not good for $x \geq 4$ but this is because when $k \geq N/4$ elements are marked the Grover search will not work. A typical way to get around this is to double the length of the list by adding N extra items which will evaluate to 0 under f.



(a) IBMQX Implementation



(b) Simulator Implementation

FIG. 34: The results of running 1000 trials on both (a) the IBMQX4 chip and (b) the IBM simulator to find values less than or equal to the input x.

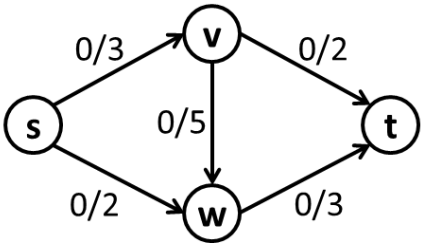


FIG. 35: A simple directed graph representing flows and capacities. Conventionally, the state of the flow problem is indicated by the current flow relative to the capacity on any directed link using the notation f/c.

D. Conclusion

While an interesting theoretical application of quantum computers, Grover's algorithm and those based upon it, such as finding minimal spanning trees, are very far from being practical, even on small toy problems. Our experience shows that when gates (or topology) is limited, then the number of gates necessary to implement generic algorithms increases significantly. (The CCX gate used here requires 15 one and two-qubit gates on the IBMX4.) Long quantum programs lead to decoherence, so this is a well known problem. However, even supposing this gets solved, quantum search still requires loading into quantum memory the objects to be searched. Thus, barring new algorithmic advances, quantum search will only replace classical search when qubits are as easy to create as classical bits.

XII. QUANTUM MAXIMUM FLOW ANALYSIS

A. Problem definition and background

Network flow problems play a major role in computational graph theory and operations research (OR). Solving the max-flow problem is the key to solving many important graph problems, such as finding a minimum cut set, and finding a maximal graph matching. The Ford-Fulkerson algorithm [40] is a landmark method that defines key heuristics for solving the max flow problem. The most important of these heuristics include the construction of a residual graph, and the notion of augmenting paths. For integer-capacity flows, Ford-Fulkerson has complexity $O(fm)$ for m edges and max flow f. The Edmonds-Karp variant has complexity $O(nm^2)$ for n vertices and m edges. The quantum-accelerated classical algorithm discussed here [4] claims complexity $O(n^{7/6}\sqrt{m})$.

The best classical implementations of the max-flow solver involve several important improvements [37], especially that of using breadth-first search to find the shortest augmenting path on each iteration. This is equivalent to constructing layered subgraphs for finding augmenting paths.

An illustration of the essential method introduced by Ford and Fulkerson can be described using Figures 35 and 36. At each link in the network, the current flow f and the capacity c are shown. Typically, the state of flow on the graph is designated c/f, with the residual capacity implicitly given by $c - f$. In Figure 35, the initial flow has been set to zero.

The basic steps in the solution to the max-flow problem are illustrated by Figure 36. The algorithm begins on the left by considering the path [s,v,t]. Since 2 is the maximum capacity allowed along that path, all the flows on the path are tacitly set to that value. Implicitly, a reverse flow of -2 is also assigned to each edge so that the tacit flow may be "undone" if necessary. Next, the middle of the figure examines the lower path [s,w,t]. This path is constrained by a maximum capacity on the edge [s,w] of again 2. Finally, the path [s,v,w,t] is the only remaining path. It can only support the residual capacity of 1 on edge [s,v]. We can then read off the maximum flow result at the sink vertex t since the total flow must end there. The maximum flow is seen to be 5.

While this method seems straightforward, without the efficiencies provided by the improvements of Edmonds and Karp, convergence might be slow for integer flows on large graphs, and may not converge at all for real-valued flows. The modification of always choosing the shortest next path in the residual network to augment is what makes the algorithm practical. To see this, consider what would have happened if the path [s,v,w,t] had been chosen first. Since aumenting that path blocks the remaining paths, flows would have to be reversed before the algorithm could proceed.

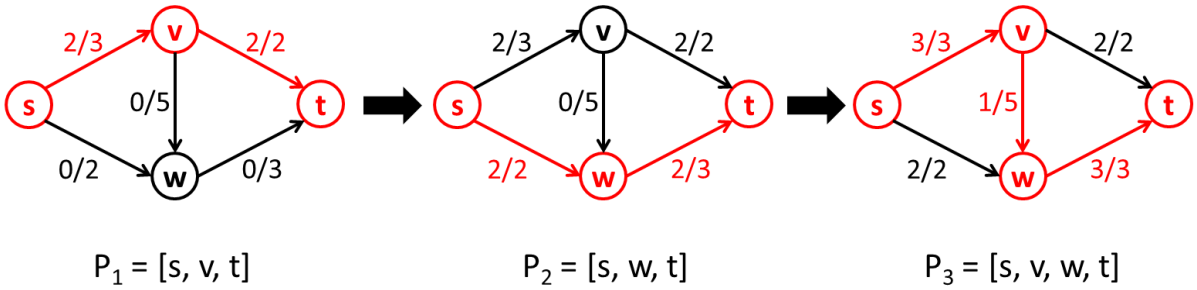


FIG. 36: The Ford-Fulkerson solution to the max-flow problem in three steps. Each step represents the application of an augmenting path to the previous flow state.

Choosing the shortest path requires performing a breadth-first search whenever new flow values have been assigned to the residual graph. This is equivalent to building a layered set of subgraphs to partition the residual graph. This is the step that leads to the m^2 complexity of Edmonds-Karp, and it is this step that is speeded up in the "quantized" version of the algorithm, leading to a complexity term of $\sqrt{m}$ instead of m^2.

B. Algorithm description

The Quantum algorithm described by Ambainis and Spalek is a "quantized" version of the Edmonds-Karp algorithm, that is, the classical algorithm with quantum acceleration. The key quantum component is a generalized version of Grover's search algorithm that finds k items in an unsorted list of length L [14]. The algorithm is used in creating a layered subgraph data structure that is subsequently used to find the shortest augmenting path at a given iteration. Another important quantum algorithm is a counting algorithm that estimates the number of ones in a bit string [15]. The counting algorithm is used in conjunction with the generallized Grover's search to estimate the number of times the search operator must be applied. The quantum speed-up comes from performing these searches quickly.

The key to "quantization" lies in using Grover's search to build a layered graph partition by computing layer numbers for all vertices. The layers are represented by an array $\mathcal{L}$ indexing the vertices of the graph, and assigning to each element a subgraph layer number. The sink vertex at vertex zero is set to zero. The the algorithm proceeds according to the following pseudo-code:

1. Set $\mathcal{L}(0) = 0$ and $\mathcal{L}(x) = \infty$ for $x \neq 0$
2. Create a one-entry queue $W = \{s\} \quad (x = 0)$
3. While $W \neq \phi$:
   (a) Take the first vertex x from W
   (b) Find by Grover search all its neighbors y with $(y) = \infty$
   (c) Set $\mathcal{L}(y) = \mathcal{L}(x) + 1$, append y into W, and remove x from W

Grover's search speeds up the layers assignment of the vertices by quickly finding all the entries in the layer array L that contain the value ∞. In practical terms, ∞ might simply be the largest value reachable in an n-qubit machine. The generalized Grover search would look for all such values, with the quantum counting algorithm determining how many Grover operations would be required to complete the search.

C. Algorithm implemented on IBM's 5-qubit computer

Since the heart of the quantized algorithm is the generalized Grover algorithm, this is the key algorithm to implement. This constituted the extent of our explorations described in this report. A circuit for Grover's algorithm was successfully implemented for three qubits and k=1, in a manner extensible to the general version of Boyer et al. Pictured in Figure 37 is the circuit. The quantum state for the results of searching for the value "six" are shown in Figure 38.

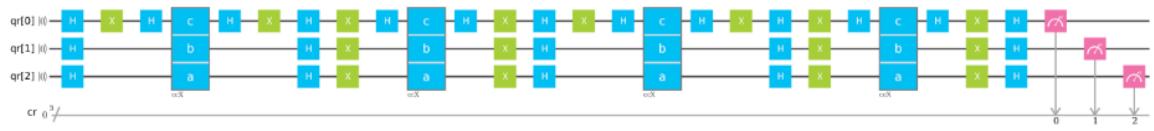


FIG. 37: The circuit that implements Grover's algorithm on the IBM 5-qubit computer.

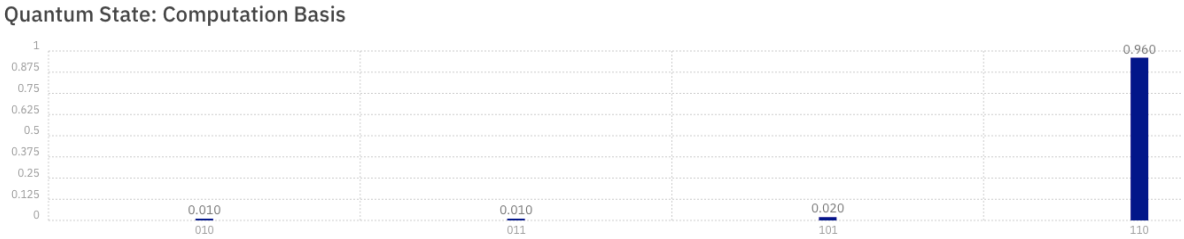


FIG. 38: The resulting state when Grover's algorithm is asked to find the number six.

D. Conclusion

In this report we described our exploration of how to implement a maximum flow solver on a quantum gate computer. The algorithm chosen for exploration is the one described by Albainis and Spalek based on "quantization" of the Edmonds-Karp algorithm. The essential quantum component is a generalized Grover search whose role is to speed up the search for augmenting paths by speeding up the assignment of vertices to subgraph layers in a breadth-first search of the available augmenting paths. In our efforts to implement the algorithm via the IBM quantum experience, we were successful in so far as implementing and running a generalizable version of Grover's search. The next step in the process is to implement the classical parts of the algorithm and couple them to the quantum computation.

XIII. QUANTUM APPROXIMATE OPTIMIZATION ALGORITHM

A. Problem Definition and Background

Combinatorial optimization problems are pervasive and appear in applications such as, hardware verification, artificial intelligence and compiler design, just to name a few. Some examples of combinatorial optimization problems include the, Knapsack, Traveling Saleman, Vehicle Routing, and Graph Coloring problems. A variety of combinatorial optimization problems including MaxSat, MaxCut, and MaxClique, can be characterized by the following generic unconstrained discrete maximization problem,

$$\begin{aligned} &\text{maximize: } \sum_{\alpha=1}^{m} C_\alpha(z) \\ &z_i \in \{0,1\}\ \forall i \in \{1,\ \ldots, n\} \end{aligned} \tag{44}$$

In this generic formulation, there are n binary decisions variables, z, and m binary functions of those variables, $C(z)$, called clauses. The challenge is to find the assignment of z that maximizes the number of clauses that can be satisfied. This is the so-called MaxSat problem, which is NP-Hard in general [62], and is an optimization variant of the well-known satisfiability problem, which is the NP-Complete [26]. Hence, solving an instantiation of (44) in practice can be computationally challenging.

To provide a concrete example of (44), let us consider the MaxCut problem. As input, the MaxCut problem takes a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, which is characterized by a set of nodes $\mathcal{V}$ and a set of undetected edges $\mathcal{E}$. The task is to partition the nodes into two sets such that, the number of edges crossing these sets is maximized. Figure 39 provides a

illustrative example. In this example, a graph with five nodes and six edges is partitioned into two sets that result in a cut of size five.

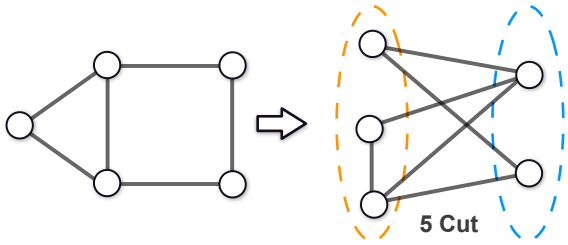


FIG. 39: An illustration of the MaxCut problem.

In general, the MaxCut problem is characterized by the following unconstrained discrete maximization problem,

$$\begin{aligned} &\text{maximize:} \sum_{i,j\in\mathcal{E}} \frac{1}{2}(1-\sigma_i\sigma_j) \\ &\sigma_i \in \{-1,1\}\ \forall i \in \mathcal{N} \end{aligned} \tag{45}$$

In this formulation, there is one binary decision variable for each node in the graph, indicating which set it belongs. The objective function consists of one term for each edge in the graph. This term is 0 if the the nodes of that edge take the same value and 1 otherwise. Consequently, the optimal solution of (45) will be a maximal cut of the graph $\mathcal{G}$. Interestingly, the form of (45) also highlights that finding a maximal cut of $\mathcal{G}$ is equivalent of find a ground state of the antiferromagnet of $\mathcal{G}$ in an Ising model interpretation. In either case, it is clear that (45) conforms to the structure of (44). Note that the following linear transform $x = (\sigma + 1)/2$, can be used to convert the variables from the $\sigma \in \{-1, 1\}$ space to the $z \in \{0, 1\}$ space.

B. Algorithm description

Throughout this section we will use the following notation for the standard quantum gates,

$$\boldsymbol{H} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \boldsymbol{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \boldsymbol{Z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

The Quantum Approximate Optimization Algorithm (QAOA) as proposed in [39] leverages gate-based quantum computing for finding high-quality solutions to combinatorial optimization problems that have the form of (44). To apply this algorithm the user first translates the clause functions $C_\alpha(z)$ into equivalent quantum Hamiltonians C_α and then selects a number of rounds $r \geq 1$ and two angles $0 \leq \beta \leq \pi$ and $0 \leq \gamma \leq 2\pi$ for each round. Starting from qubits in the $|0 \ldots 0\rangle$ state, the following quantum operation is applied,

$$\prod_{j=1}^{n} \boldsymbol{H}_j \prod_{k=1}^{r} \left(\prod_{j=1}^{n} \boldsymbol{e}^{-\boldsymbol{i}\boldsymbol{\beta}_k \boldsymbol{X}_j} \prod_{\alpha=1}^{m} \boldsymbol{e}^{-\boldsymbol{i}\boldsymbol{\gamma}_k \boldsymbol{C}_\alpha} \right) \tag{46}$$

and, for an appropriate selection of r, β, γ, the state of the qubits after this transformation will be a high-quality solution to (44), with high probability. As the number of rounds used increases, the quality of the solution produced by (46) also increases. Conducting an exhaustive search over a fine grid is proposed in [39] for the selection of each round's β, γ parameters. The use of an quantum-variational-eigensolver is also possible [72, 80].

To provide a concrete example of the generic QAOA formulation (46), let us consider its application to the MaxCut problem (45). It is important to note that the MaxCut problem is particularly advantageous for QAOA for the following reasons: (1) all of the clauses in the objective function have the same structure, hence only one quantum Hamiltonian C_α needs to be designed; (2) The clauses only involve two decision variables, which keeps the structure of

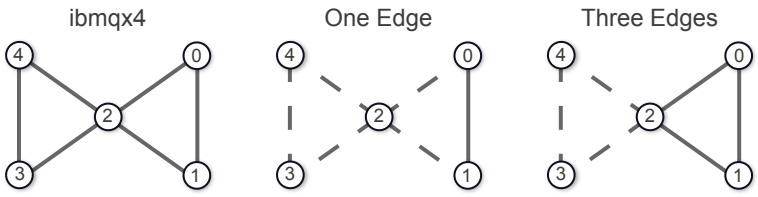


FIG. 40: The CNOT connectivity of the IBMQX4 Computer (left) followed by the one edge (center) and three edge (right) graphs considered in the proof-of-concept experiments.

quantum Hamiltonian C_α relativity simple. The design of MaxCut quantum Hamiltonian is as follows,

$$C(i,j) = \frac{1}{2}(1 - \sigma_i\sigma_j) \tag{47}$$

$$\begin{array}{c} \sigma_i = -1 \;\; \sigma_i = 1 \\ \left(\begin{array}{cc} 0 & 1 \\ 1 & 0 \end{array} \right) \begin{array}{l} \sigma_j = -1 \\ \sigma_j = 1 \end{array} \end{array} \tag{48}$$

$$\begin{array}{c} |00\rangle \;\; |01\rangle \;\; |10\rangle \;\; |11\rangle \\ \left(\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right) \begin{array}{l} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{array} \end{array} \tag{49}$$

$$C_{i,j} = \frac{1}{2}(\boldsymbol{I} - \boldsymbol{Z}_i \otimes \boldsymbol{Z}_j) \tag{50}$$

The first expression (47) presents the binary function used by each edge in the objective of (45). The second expression (48) shows the enumeration of all inputs and outputs of the binary function, and (49) illustrates how to encode these inputs and outputs into a quantum Hamiltonian. Finally, the quantum Hamiltonian (49) can be compactly written as (50). Combining (46) with (50) and the input graph, $\mathcal{G}$, yields the complete QAOA MaxCut Hamiltonian,

$$\prod_{j\in\mathcal{N}} \boldsymbol{H}_j \prod_{k=1}^{r} \left(\prod_{j\in\mathcal{N}} \boldsymbol{e}^{-\boldsymbol{i}\beta_k \boldsymbol{X}_j} \prod_{a,b\in\mathcal{E}} \boldsymbol{e}^{-\boldsymbol{i}\frac{\gamma_k}{2}(\boldsymbol{I}-\boldsymbol{Z}_a\otimes\boldsymbol{Z}_b)} \right) \tag{51}$$

C. QAOA MaxCut on the IBMQX4 Quantum Computer

This section investigates the implementation of QAOA MaxCut Hamiltonian (51) on the IBMQX4 quantum computer (Figure 40), which is accessible as part of the IBM Quantum Experience [51]. The first challenge is to transform (51) from its mathematical form into a sequence of operations that are available in the IBM Quantum Experience platform. To that end, the following unitary gates are used,

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad U1(\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & \boldsymbol{e}^{\boldsymbol{i}\lambda} \end{pmatrix} \quad U3(\theta,\phi,\lambda) = \begin{pmatrix} \cos(\theta/2) & -\boldsymbol{e}^{\boldsymbol{i}\lambda}\sin(\theta/2) \\ \boldsymbol{e}^{\boldsymbol{i}\phi}\sin(\theta/2) & \boldsymbol{e}^{\boldsymbol{i}\lambda+\phi}\cos(\theta/2) \end{pmatrix},$$

as well as the two-qubit controlled NOT gate,

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

The first collection of terms in (51) are the pervasive Hadamard gates and are implemented via the H gate on IBM's

interface. The second collection of terms is the application of the β angle, which are expanded as follows,

$$e^{-i\beta_k X} = e^{-i\beta_k \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}} = \begin{pmatrix} \cos(\beta_k) & -i\sin(\beta_k) \\ i\sin(\beta_k) & \cos(\beta_k) \end{pmatrix} \tag{52}$$

Careful inspection of the IBM Quantum Experience gates reveals that (52) is implemented by $U3(2\beta_k, -\pi/2, \pi/2)$. The third collection of terms is the application of the γ angle and MaxCut Hamiltonian, which are expanded as follows,

$$e^{-i\frac{\gamma_k}{2}(I - Z_a \otimes Z_b)} = e^{-i\frac{\gamma_k}{2}\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-i\frac{\gamma_k}{2}} & 0 & 0 \\ 0 & 0 & e^{-i\frac{\gamma_k}{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{53}$$

After some derivation, it is observed that this gate can be implemented as a sequence of X, $U1(-\gamma/2)$, and CNOT gates as indicated in Figure 41.[1]

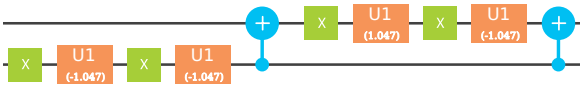


FIG. 41: An IBM Quantum Experience score illustrating an implementation of the MaxCut edge gate (53).

Putting all of these components together Figure 42 presents an IBM Quantum Experience score for solving the QAOA MaxCut Hamiltonian (51) on a one edge graph using the following parameters,

$$\begin{aligned} r &= 2 \\ \beta_1 &= 0.000, \gamma_1 = 2.094 \\ \beta_2 &= 2.094, \gamma_2 = 2.094 \end{aligned}$$

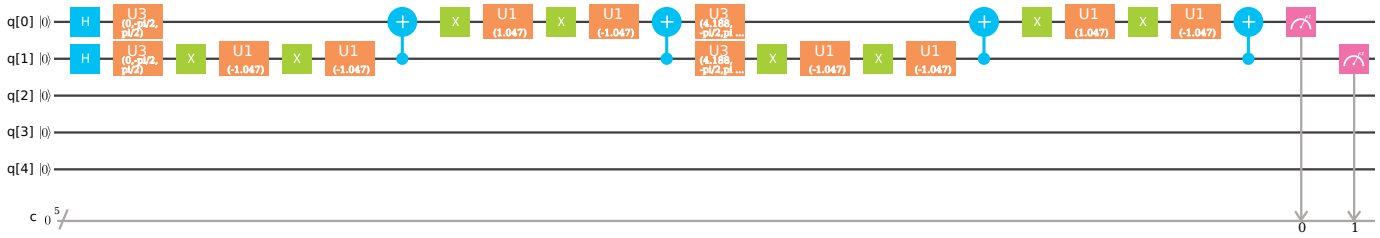


FIG. 42: An IBM Quantum Experience score for QAOA MaxCut with two rounds on a one edge graph.

D. A Proof-of-Concept Experiment

With a basic implementation of the QAOA MaxCut Hamiltonian (51) in the IBM Quantum Experience, a preliminary proof-of-concept study is conducted to investigate the effectiveness of QAOA MaxCut for finding high-quality cuts in the one edge and three edge graphs presented in Figure 40. This study compares three types of MaxCut computations: (1) *Random* assigns each node in the graph to one of the two sets uniformly at random; (2) *Simulation* executes the IBM Quantum Experience score via simulation on a classical computer; (3) *Hardware* executes the IBM Quantum Experience score on the IBMQX4 hardware. The simulation computation serves to demonstrate the mathematical

[1] It is interesting to note the alternate implementation of this gate in [25], which leverages a wider variety of gate operations [97].

correctness of the proposed QAOA score. The hardware computation demonstrates the viability of the score in a deployment scenario where environmental noise, intrinsic bias, and decoherence can have a significant impact on the results. The random computation serves to demonstrate that the hardware results are better than what one would expect by chance from pure noise. All of these computations are stochastic, therefore event probabilities are computed based on 1024 replicates of each computation.

The first experiment considers the one edge graph from Figure 40 and implements (51) with the following parameters,

$$\begin{aligned} \boldsymbol{r} &= 2 \\ \boldsymbol{\beta}_1 = 0.0785, \boldsymbol{\gamma}_1 &= 1.4137 \\ \boldsymbol{\beta}_2 = 0.3927, \boldsymbol{\gamma}_2 &= 0.6283 \end{aligned}$$

The results are summarized in Table VII. The simulation results indicate that the proposed score is mathematically sound and the hardware results indicate similar performance to the simulation, with a few additional errors. The random results indicate that both the simulation and hardware perform significantly better than random chance.

TABLE VII: Cut size distributions of QAOA MaxCut with two rounds on a one edge graph.

Cut Size	Random	Simulation	Hardware
1	0.50	0.99	0.91
0	0.50	0.01	0.09

The second experiment considers the three edge graph from Figure 40 and implements (51) with the following parameters,

$$\begin{aligned} \boldsymbol{r} &= 2 \\ \boldsymbol{\beta}_1 = 0.0785, \boldsymbol{\gamma}_1 &= 0.6283 \\ \boldsymbol{\beta}_2 = 0.3142, \boldsymbol{\gamma}_2 &= 0.1571 \end{aligned}$$

The results are summarized in Table VIII. The simulation results indicate that the proposed score is mathematically sound. The hardware results indicate some degradation in performance when compared to simulation. However, the random results indicate that the hardware still performs better than random chance. It is worthwhile to note that the score required to implement this experiment is around 70 operations and at that length, environmental noise and qubit decoherence may be a significant factor.

TABLE VIII: Cut size distributions of QAOA MaxCut with two rounds on a three edge graph.

Cut Size	Random	Simulation	Hardware
2	0.75	1.00	0.84
0	0.25	0.00	0.16

E. Conclusion

This work considered the the application QAOA [39] for the MaxCut problem on the IBMQX4 quantum computer. Simulated results demonstrate the correctness of the QAOA MaxCut implementation and experimental results on IBMQX4 hardware showed that similar results can be achieved for graphs with up to three nodes and three edges.

XIV. QUANTUM PRINCIPAL COMPONENT ANALYSIS

A. Problem definition and background

In data analysis, it is common to have many features, some of which are redundant or correlated. As an example, consider housing prices, which are a function of many features of the house, such as the number of bedrooms, number of bathrooms, square footage, lot size, date of construction, and the location of the house. Often, one is interested in reducing the number of features to the few, most important features. Here, by important, we mean features that

capture the largest variance in the data. For example, if one is only considering houses on one particular street, then the location may not be important, while the square footage may capture a large variance.

Determining which features capture the largest variance is the goal of Principal Component Anaylsis (PCA) [79]. Mathematically, PCA involves taking the raw data (e.g., the feature vectors for various houses) and computing the covariance matrix, Σ. For example, for two features, X_1 and X_2, the covariance is given by

$$\Sigma = \begin{pmatrix} \mathbf{E}(X_1 * X_1) & \mathbf{E}(X_1 * X_2) \\ \mathbf{E}(X_2 * X_1) & \mathbf{E}(X_2 * X_2) \end{pmatrix} , \tag{54}$$

where $\mathbf{E}(A)$ is the expectation value of A, and we have assumed that $\mathbf{E}(X_1) = \mathbf{E}(X_2)$=0. Next, one diagonalizes Σ such that the eigenvalues $e_1 \geq e_2 \geq e_3 \geq \cdots$ are listed in decreasing order. Again, for the two-feature case, this becomes

$$\Sigma = \begin{pmatrix} e_1 & 0 \\ 0 & e_2 \end{pmatrix} . \tag{55}$$

Once Σ is in this form, one can choose to keep the features with n-largest eigenvalues and discard the other features. Here, n is a free parameter that depends on how much one wants to reduce the dimensionality. Naturally, if there are only two features, one would consider $n = 1$, i.e., the single feature that captures the largest variance.

As an example, consider the number of bedrooms and the square footage of several houses for sale in Los Alamos. Here is the raw data, taken from www.zillow.com, for 15 houses:

$$\begin{aligned} X_1 &= \text{number of bedrooms} = \{4, 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 5, 4, 3, 4\} \\ X_2 &= \text{square footage} = \{3028, 1365, 2726, 2538, 1318, 1693, 1412, 1632, 2875, 3564, 4412, 4444, 4278, 3064, 3857\} . \end{aligned} \tag{56}$$

Henceforth, for scaling purposes, we will divide the square footage by 1000 and subtract off the mean of both features. Classically, we compute the covariance matrix and its eigenvalues to be the following:

$$\Sigma = \begin{pmatrix} 0.380952 & 0.573476 \\ 0.573476 & 1.29693 \end{pmatrix} , \quad e_1 = 1.57286 , \quad e_2 = 0.105029 . \tag{57}$$

We now discuss the quantum algorithm for doing the above calculation, i.e., for finding the eigenvalues of Σ.

B. Algorithm description

A quantum algorithm for performing PCA was presented in Ref. [68]. The algorithm: (1) encodes Σ in a quantum density matrix ρ (exploiting the fact that Σ is a positive semidefinite matrix), (2) prepares many copies of ρ, (3) performs the exponential SWAP operation on each copy and a target system, and (4) performs quantum phase estimation to determine the eigenvalues. However, given the constraint of only 5 qubits on IBM's computer, preparing many copies of ρ is not possible. Hence, we consider a simpler algorithm as follows.

In the special case where there are only two features, Σ and ρ are 2×2 matrices, and ρ can be purified to a pure state $|\psi\rangle$ on two qubits. Suppose one prepares two copies of $|\psi\rangle$, which uses a total of 4 qubits, then the fifth qubit (on IBM's computer) can be used an ancilla to implement an algorithm that determines the purity $P := \mathrm{Tr}(\rho^2)$ of ρ. This algorithm was discussed, e.g., in Ref. [56]. It is then straightforward to calculate the eigenvalues of Σ from P, as follows:

$$e_1 = \mathrm{Tr}(\Sigma) * (1 + \sqrt{1 - 2(1 - P)})/2 \tag{58}$$

$$e_2 = \mathrm{Tr}(\Sigma) * (1 - \sqrt{1 - 2(1 - P)})/2 . \tag{59}$$

As depicted in Fig. 43, this simple algorithm is schematically divided up into four steps: (1) classical pre-processing, (2) state preparation, (3) quantifying the purity, and (4) classical post-processing.

In the first step, one's classical computer converts the raw data vectors into a covariance matrix Σ, then normalizes this matrix to form $\rho = \Sigma/\mathrm{Tr}(\Sigma)$, then purifies it to make a pure state $|\psi\rangle$, and finally computes the unitary U_{prep} needed to prepare $|\psi\rangle$ from a pair of qubits each initially in the $|0\rangle$ state.

In the second step, one's quantum computer actually prepares the state $|\psi\rangle$, or in fact, two copies of $|\psi\rangle$, using U_{prep}, which can be decomposed as follows:

$$U_{\mathrm{prep}} = (U_A \otimes U_B)\mathrm{CNOT}_{AB}(U'_A \otimes \mathbb{1}_B) . \tag{60}$$

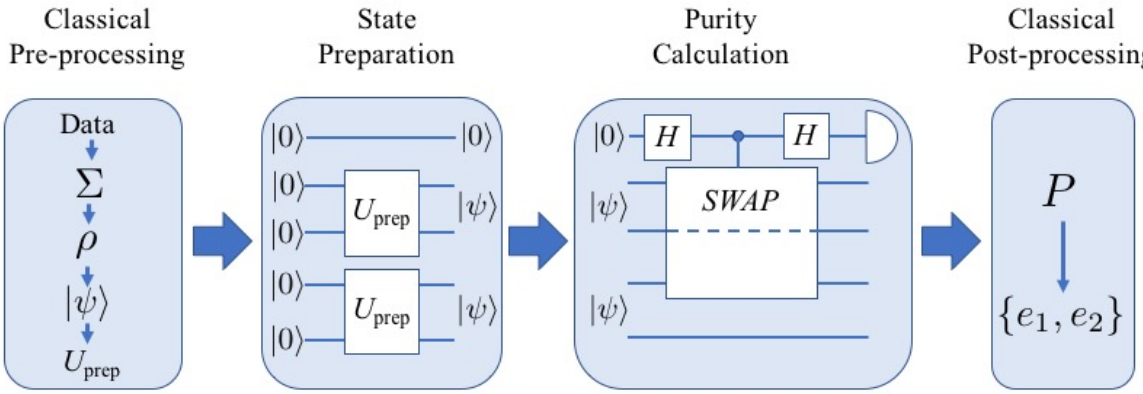


FIG. 43: Schematic diagram for the quantum algorithm for PCA, in the special case of only two features. The first step is classical pre-processing: transforming the raw data into a covariance matrix Σ, then normalizing to compute $\rho = \Sigma/\mathrm{Tr}(\Sigma)$, then purifying ρ to a two-qubit pure state $|\psi\rangle$, and finally determining the unitary $U_{\rm prep}$ needed to prepare $|\psi\rangle$. The second step is to prepare two copies of $|\psi\rangle$ by implementing $U_{\rm prep}$ on a quantum computer. The third step is purity calculation, which is the bulk of the quantum algorithm. This involves doing a Hadamard on an ancilla, which then is used to implement a controlled-SWAP gate on two qubits (from different copies of $|\psi\rangle$), and then another Hadamard on the ancilla. Finally measuring $\langle Z\rangle$ on the ancilla gives the purity $P = \mathrm{Tr}(\rho^2)$. The last step is to classically compute the eigenvalues using Eqs. (58)-(59).

Note that $U_{\rm prep}$ acts on two qubits, denoted A and B, and CNOT_{AB} is a CNOT gate with A the control qubit and B the target. The single qubit unitaries U_A, U_B, and U'_A can be written in IBM's standard form:

$$\begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i\lambda+\phi}\cos(\theta/2) \end{pmatrix}, \tag{61}$$

where the parameters θ, λ, and ϕ were calculated in the previous (classical pre-processing) step.

The third step is purity calculation, which makes up the bulk of the quantum algorithm. As shown in Fig. 43, first one does a Hadamard on an ancilla. Let us denote the ancilla as C, while the other four qubits are denoted A, B, A', and B'. During the state preparation step, qubits A and B were prepared in state $|\psi\rangle$ with the reduced state being $\rho_A = \mathrm{Tr}_B(|\psi\rangle\langle\psi|) = \rho$. Likewise we have $\rho_{A'} = \mathrm{Tr}_{B'}(|\psi\rangle\langle\psi|) = \rho$. Next, qubit C is used to control a controlled-SWAP gate, where the targets of the controlled-SWAP are qubits A and A'. Then, another Hadamard is performed on C. Finally, C is measured in the Z basis. One can show that the final expectation value of Z on qubit C is precisely the purity of ρ, i.e.,

$$\langle Z\rangle_C = p_0 - p_1 = \mathrm{Tr}(\rho^2) = P\,, \tag{62}$$

where p_0 (p_1) is the probability for the zero (one) outcome on C.

The fourth step is classical post-processing, where one converts P into the eigenvalues of Σ using Eqs. (58) and (59).

C. Algorithm implemented on IBM's 5-qubit computer

The actual gate sequence that we implemented on IBM's 5-qubit computer is shown in Fig. 44. This involved a total of 16 CNOT gates. The decomposition of controlled-SWAP into one- and two-qubit gates is done first by relating it to the Toffoli gate:

$$\text{controlled-SWAP}_{CAB} = (\mathbb{1}_C \otimes \mathrm{CNOT}_{BA})\mathrm{Toffoli}_{CAB}(\mathbb{1}_C \otimes \mathrm{CNOT}_{BA}) \tag{63}$$

and then decomposing the Toffoli gate, as in Ref. [92].

We note that the limited connectivity of IBM's computer played a signficant role in determining the algorithm. For example, we needed to implement a CNOT from q[1] to q[2], which required a circuit that reverses the direction of the CNOT from q[2] to q[1]. Also, we needed a CNOT from q[3] to q[1], which required a circuit involving a total of four CNOTs (from q[3] to q[2] and from q[2] to q[1]).

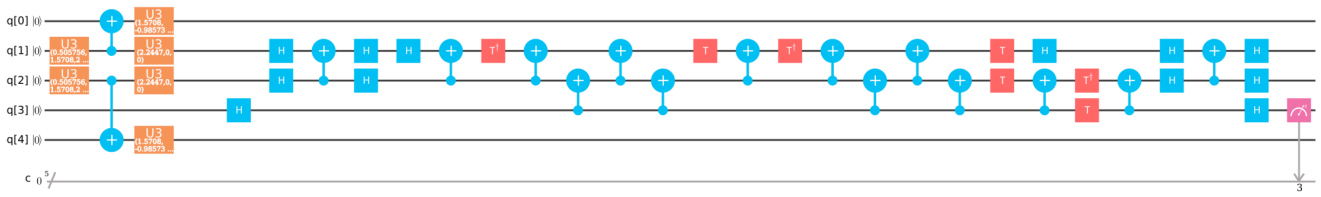


FIG. 44: Actual circuit implemented on IBM's 5-qubit simulator and quantum computer. The first three time slots in the score correspond to the state preparation step of the algorithm, and the subsequent time slots correspond to the purity calculation step. Due to connectivity reasons, we chose qubit q[3] as the ancilla and qubits q[1] and q[2] as the targets of the controlled-SWAP operation. We decomposed the controlled-SWAP operation into CNOT gates by first relating it to the Toffoli gate via Eq. (63), and then decomposing the Toffoli gate into CNOT gates [92].

Our results are as follows. For the example given in Eq. (56), IBM's 5-qubit simulator with 40960 trials gave:

$$e_1 = 1.57492\,, \quad e_2 = 0.102965 \quad \text{(IBM's simulator)}\,. \tag{64}$$

A comparison with Eq. (57) shows that IBM's simulator essentially gave the correct answer. On the other hand, IBM's 5-qubit quantum computer with 40960 trials gave:

$$e_1 = 0.838943 + 0.45396i\,, \quad e_2 = 0.838943 - 0.45396i \quad \text{(IBM's Quantum Computer)}\,. \tag{65}$$

This is a non-sensical result, since the eigenvalues of a covariance matrix must be (non-negative) real numbers. So, unfortunately IBM's quantum computer did not give the correct answer.

D. Conclusion

In conclusion, we implemented a quantum algorithm for performing PCA on a two-feature data set. IBM's 5-qubit simulator gave the correct answer for the eigenvalues of the covariance matrix. However, IBM's 5-qubit quantum computer did not give a sensible result, giving non-real eigenvalues. We hypothesize that decoherence played a role in the quantum computation. Because our algorithm in Fig. 44 is fairly long, with 16 CNOT gates, partial decoherence likely occurred during this algorithm. This is consistent with the probability distribution for the final measurement being more like that of a mixed state.

XV. QUANTUM SUPPORT VECTOR MACHINE

A. Quantum Support Vector Machine

Support Vector Machines (SVM) are a class of supervised machine learning algorithms for binary classifications. Consider M data points of $\{(\vec{x}_j, y_j) : j = 1, 2, \ldots, M\}$. Here $\vec{x}_j$ is a N-dimensional vector in data feature space, and y_j is the label of the data, which is $+1$ or -1. SVM finds the hyperplane $\vec{w} \cdot \vec{x} + b = 0$ that divides the data points into two categories so that $\vec{w} \cdot \vec{x}_j + b \geq 1$ when $y_j = +1$ and $\vec{w} \cdot \vec{x}_j + b \leq -1$ when $y_j = -1$, and that is maximally separated from the nearest data points on each category. Least Squares SVM (LS-SVM) is a version of SVM [101]. It approximates the hyperplane finding procedure of SVM by solving the following linear equation:

$$\begin{bmatrix} 0 & \vec{1}^T \\ \vec{1} & \mathbf{K} + \gamma^{-1}\mathbf{1} \end{bmatrix} \begin{bmatrix} b \\ \vec{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \vec{y} \end{bmatrix}\,. \tag{66}$$

Here $\mathbf{K}$ is called the kernel matrix of dimension $M \times M$, γ is a tuning parameter, and $\vec{\alpha}$ forms the normal vector $\vec{w}$ by $\vec{w} = \sum_{j=1}^{M} \alpha_j \vec{x}_j$. Various definitions for the kernel matrix are available, but the quantum SVM [85] uses linear kernel: $K_{ij} = \vec{x}_i \cdot \vec{x}_j$. Classically, the complexity of the LS-SVM is $\mathcal{O}\big(M^2(M+N)\big)$.

Quantum version of SVM performs the LS-SVM algorithm using quantum computers [85]. It calculates the kernel matrix using the quantum algorithm for inner product [66] on quantum random access memory [46], solves the linear

equation using a quantum algorithm for solving linear equations [46], and perform the classification of a query data using the trained qubits with a quantum algorithm [85]. The overall complexity of the quantum SVM is $\mathcal{O}\big(\log NM\big)$. In this letter, we will focus only on the quantum algorithm for solving linear equations.

A quantum version of SVM performs the LS-SVM algorithm using quantum computers [85]. It calculates the kernel matrix using the quantum algorithm for inner product [66] on quantum random access memory [46], solves the linear equation using a quantum algorithm for solving linear equations [46], and perform the classification of a query data using the trained qubits with a quantum algorithm [85]. The algorithm can be summarized as below:

Algorithm 3 Quantum SVM [85]

Input:
- Training data set $\{(\vec{x}_j, y_j) : j = 1, 2, \ldots, M\}$.
- A query data $\vec{x}$.

Output:
- Classification of $\vec{x}$: $+1$ or -1.

Procedure:

Step 1. Calculate kernel matrix $K_{ij} = \vec{x}_i \cdot \vec{x}_j$ using quantum inner product algorithm [66].

Step 2. Solve linear equation Eq. (66) and find $|b, \vec{\alpha}\rangle$ using a quantum algorithm for solving linear equations [46] (training step).

Step 3. Perform classification of the query data $\vec{x}$ against the training results $|b, \vec{\alpha}\rangle$ using a quantum algorithm [85].

The overall complexity of the quantum SVM is $\mathcal{O}\big(\log NM\big)$. In this section, we will focus only on the quantum algorithm for solving linear equations.

B. Quantum Algorithm for Solving Linear Equations

Classically, the problem of linear equations is to find an unknown vector $\vec{x}$ in the following linear equation: $A\vec{x} = \vec{b}$ for a known $N \times N$ matrix A, and a known N-dimensional vector $\vec{b}$. The quantum version of the problem can be written as $A|x\rangle = |b\rangle$, where A is Hermitian, and $|x\rangle$ and $|b\rangle$ are the quantum states representing the $\vec{x}$ and $\vec{b}$ vectors. The quantum states can hold only the normalized vectors, so the solution we will find is $|x\rangle = A^{-1}|b\rangle/||A^{-1}|b\rangle||$. If the matrix A is not Hermitian, one can solve $\begin{bmatrix} 0 & A \\ A^\dagger & 0 \end{bmatrix} \begin{bmatrix} 0 \\ \vec{x} \end{bmatrix} = \begin{bmatrix} \vec{b} \\ 0 \end{bmatrix}$, instead of the original equation, so the algorithm is general for any invertible matrices.

The matrix A and the states $|x\rangle$ and $|b\rangle$ can be expanded in terms of the eigenstates of A as

$$A = \sum_{j=1}^{N} \lambda_j |u_j\rangle\langle u_j|, \qquad A^{-1} = \sum_{j=1}^{N} \lambda_j^{-1} |u_j\rangle\langle u_j|, \tag{67}$$

$$|b\rangle = \sum_{j=1}^{N} \beta_j |u_j\rangle \quad \text{where } \beta_j = \langle u_j|b\rangle, \text{ and} \tag{68}$$

$$|x\rangle \propto A^{-1}|b\rangle = \left(\sum_{k=1}^{N} \lambda_k^{-1} |u_k\rangle\langle u_k|\right) \left(\sum_{j=1}^{N} \beta_j |u_j\rangle\right) = \sum_{j=1}^{N} \frac{\beta_j}{\lambda_j} |u_j\rangle\,, \tag{69}$$

where λ_j and $|u_j\rangle$ are the eigenvalues and eigenstates of A. Using these notations, the quantum algorithm for solving the linear equation can be described as follows.

(1) Prepare n register qubits in $|0\rangle^{\otimes n}$ state with the vector qubit $|b\rangle$:

$$|b\rangle|0\rangle^{\otimes n} = \sum_{j=1}^{N} \beta_j |u_j\rangle|0\rangle^{\otimes n}\,. \tag{70}$$

(2) Perform phase estimation for A with the register qubits $|0\rangle^{\otimes n}$ so that the eigenvalue of A corresponding to the eigenstate $|u_j\rangle$ can be stored in the register qubits:

$$\sum_{j=1}^{N} \beta_j |u_j\rangle|\lambda_j\rangle\,. \tag{71}$$

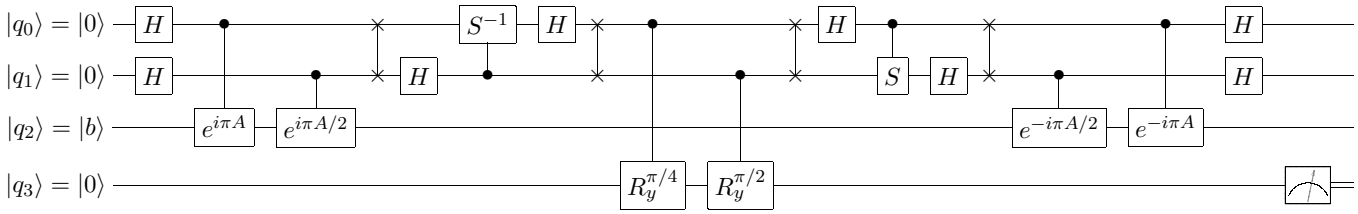


FIG. 45: Quantum circuit for solving linear equations proposed in Ref. [78]. The circuit assumes that the eigenvalues of A are $\lambda_j = \{1, 2\}$, so that reciprocals of the eigenvalues can be obtained by swap a swap gate, as explained in text. After completion of the run, $|q_2\rangle$ is the solution state of the linear equation only when the ancillary qubit $|q_3\rangle = |1\rangle$.

(3) Introduce an ancillary qubit $|0\rangle$, and perform the R_y rotation on the ancilla. The total rotation angle will be determined by the eigenvalue stored in the register qubit such that

$$\sum_{j=1}^{N} \beta_j |u_j\rangle |\lambda_j\rangle |0\rangle \quad \Longrightarrow \quad \sum_{j=1}^{N} \beta_j |u_j\rangle |\lambda_j\rangle \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \frac{C}{\lambda_j} |1\rangle \right) . \tag{72}$$

In general, the procedure requires the calculation of reciprocals of the eigenvalues λ_j^{-1}. A quantum algorithm for this calculation is proposed in Ref. [19].

(4) Perform inverse phase estimation to disentangle the eigenvalue register:

$$\sum_{j=1}^{N} \beta_j |u_j\rangle \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \frac{C}{\lambda_j} |1\rangle \right) . \tag{73}$$

(5) Postselect for the ancillary qubit of $|1\rangle$:

$$\sum_{j=1}^{N} \beta_j |u_j\rangle \frac{C}{\lambda_j} |1\rangle \propto |x\rangle . \tag{74}$$

By comparing with Eq. (69), one finds that the result is proportional to the solution state that we want to obtain.

The complexity of the classical algorithms for solving the linear equation is $\mathcal{O}(N^3)$ if A is dense, and $\mathcal{O}(Ns\kappa)$ if A is sparse with sparsity s and condition number κ. The complexity of the quantum algorithm is $\mathcal{O}(\kappa^2 \log N)$, or $\mathcal{O}(\kappa \log^3 \kappa \log N)$ with the algorithm proposed in Ref. [2].

C. Realization of the Quantum Algorithm for Solving Linear Equations

The smallest nontrivial circuit design is proposed in Ref. [78]; the quantum circuit design is shown in Fig. 45. The circuit assumes that the eigenvalues of the matrix A are $\lambda_j = \{1, 2\}$, so that the reciprocals of the eigenvalues can be calculated by a simple swap gate. Note that the two eigenvalues will be stored in two qubits as $|01\rangle$ and $|10\rangle$. By swapping them, $\lambda_1 = 1 \to 2 = 2 \times \lambda_1^{-1}$, and $\lambda_2 = 2 \to 1 = 2 \times \lambda_2^{-1}$. Hence, the swap operation will transform the eigenvalue qubits to $|\lambda_j\rangle \to |2\lambda_j^{-1}\rangle$. By performing controlled R_y-rotations of angle $\pi/4$ and $\pi/2$, the ancillary qubit $|q_3\rangle$ will be rotated by $\theta_j = \lambda_j^{-1}\pi/4$. Approximating $\sin(\theta_j/2) \approx \theta_j/2$, the rotated state becomes Eq. (72).

For the implementation of the circuit on the QISKit, the matrix A is chosen to be

$$A = \frac{1}{2} \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} = \left(R_y^{\pi/2} \right)^\dagger \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} R_y^{\pi/2} . \tag{75}$$

The phase estimation requires controlled Hamiltonian simulation of $e^{i\pi A}$ and $e^{i\pi A/2}$. Current version of QISKit does not support a simulation of arbitrary matrix, but $e^{i\pi A}$ is a simple X-gate. The controlled $e^{i\pi A/2}$ operation cannot be expressed in terms of the gates QISKit provides, so we use following approximation:

$$e^{i\pi A/2} = \frac{1}{2} \begin{bmatrix} -1+i & -1-i \\ -1-i & -1+i \end{bmatrix} \approx \left(R_y^{-\pi/2} \right)^\dagger \left(R_x^{\pi} \right)^\dagger R_z^{\pi/2} R_x^{\pi} R_y^{\pi/2} = \frac{1}{2} \begin{bmatrix} -1-i & -1-i \\ -1-i & +1-i \end{bmatrix} . \tag{76}$$

$\vec{b}$	$\vec{x}^{\text{norm}}_{\text{theory}}$	$\vec{x}^{\text{norm}}_{\text{measured}}$
$(+0.707, -0.707)$	$(+0.707, -0.707)$	$(+0.710, -0.704)$
$(+0.866, -0.500)$	$(+0.795, -0.607)$	$(+0.806, -0.592)$
$(+0.588, +0.809)$	$(+0.461, +0.888)$	$(+0.514, +0.858)$
$(-1.000, +0.000)$	$(-0.949, +0.316)$	$(-0.961, +0.277)$

TABLE IX: Expected and measured solution of the linear equation for various $\vec{b}$ vectors. The measured results are average over 10000 shots on QISKit simulator. Solution vectors are normalized by $\vec{x}^{\text{norm}} = \vec{x}/||\vec{x}||$.

In this specific example, one could avoid such approximation by designing the phase estimation circuit tailored for the eigenvectors of A as explained in Ref. [18], but here we keep the circuit and use approximation for generality.

Table IX shows the results of the quantum algorithm for solving linear equations for various $\vec{b}$ vectors. Despite the approximation made in Eq. (76), the simulation results are very close to the true solution. Unfortunately, the quantum circuit could not be implemented on the IBM Q Experience quantum computers because of the restricted connectivity of the IBM Q Experience qubits.

D. Conclusion

In this section, we discussed the quantum algorithm for solving linear equations, the key algorithm composing the quantum SVM. The algorithm is realized on the QISKit simulator using four qubits for the matrix Eq. (75). The simulation results for various $\vec{b}$ vectors are given in Table IX, and they are close to the true solution. The algorithm could not be implemented on the IBM Q Experience because of the restricted qubit connectivity.

XVI. QUANTUM SIMULATION OF THE SCHRÖDINGER EQUATION

A. Problem definition and background

The Schrödinger's equation describes evolution of a wave function $\psi(x,t)$ for a given Hamiltonian $\hat{H}$ of a quantum system:

$$i\hbar\frac{\partial}{\partial t}\psi(x,t) = \hat{H}\psi(x,t) = \left[\frac{\hbar^2\hat{k}^2}{2m} + V(\hat{x})\right]\psi(x,t), \tag{77}$$

where the second equality illustrates the Hamiltonian of a particle of mass m in a potential $V(x)$. Simulating this equation starting with a known wave function $\psi(x,0)$ provides knowledge about the wave function at a given time t_f and allows determination of observation outcomes. For example, $|\psi(x,t_f)|^2$ is the probability of finding a quantum particle at a position x after time t_f.

Solving the Schrödinger's equation numerically is the common approach since analytical solutions are known for a only handful of systems. On a classical computer, a numerical algorithm starts by defining a wave function on a discrete grid $\psi(x_i,0)$ with large number of points $i \in [1,N]$. The form of the Hamiltonian, Eq. 77, allows one to split the system's evolution on a single time step Δt in two steps, which are easy to perform:

$$\psi(x_i,t_{n+1}) = e^{-iV(x_i)\Delta t}e^{-i\hat{k}^2\Delta t}\psi(x_i,t_n), \tag{78}$$

where we have assumed that $\hbar = m = 1$. The quantum state evolution thus consists of alternating application of the phase shift operators in the coordinate and momentum representations. These two representation are linked together by the Fourier Transformation as in the following example of a free space evolution of a quantum particle:

$$\psi(x_i,t_f) = IFFT\, e^{-i\hat{k}^2 t_f}\ FFT\, \psi(x_i,0), \tag{79}$$

where $V(x) = 0$ for a free particle, and FFT and $IFFT$ are the Fast Fourier Transform and its inverse.

We now discuss the quantum simulation of the Schrödinger's equation similar to the one discussed by Somma in [99] that provides the wave function of the system at a given time t_f. Finding a proper measurement on a quantum simulator that reveals information about the quantum system will however be left out of the discussion. $|\psi(x,t_f)|^2$ will be the only information, we will be interested in finding out.

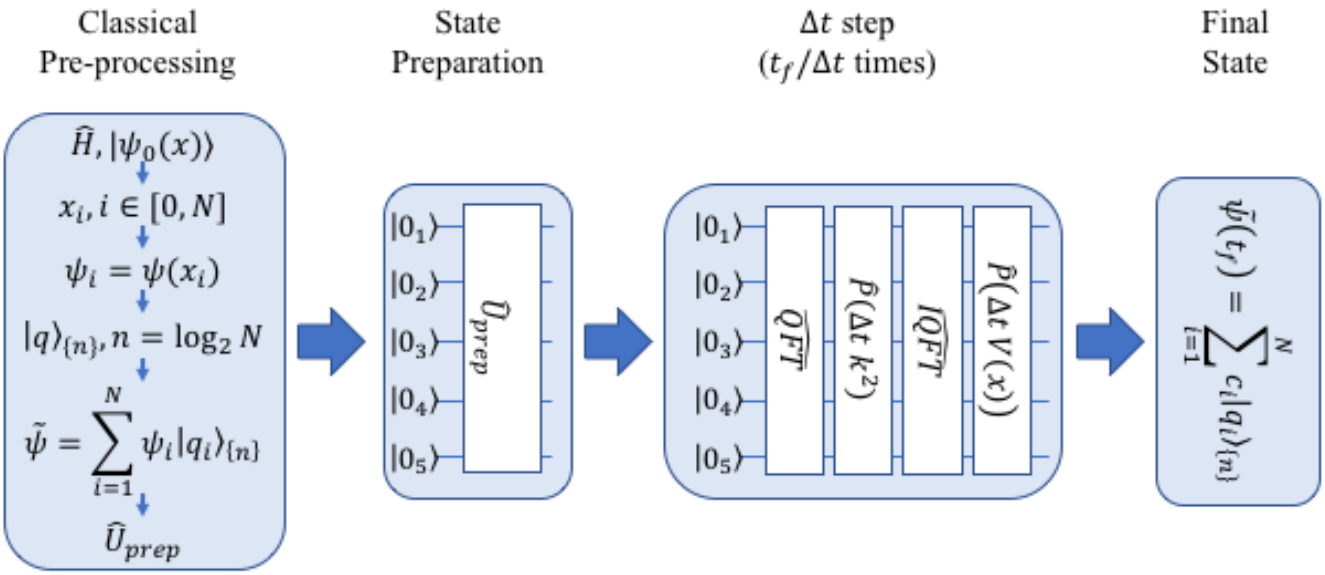


FIG. 46: The quantum simulation of the Schrödinger's equation. The first stage is classical pre-processing that encodes the wave function to available qubits and derives a state preparation operator that takes an all-zero state of a quantum computer to a desired state. The second stage prepares an initial state by implementing the state preparation operator $\hat{U}_{\text{prep}}$ on a quantum computer. The third stage is an iterative update looped over Δt steps based on the operator splitting method.

B. Algorithm description

A quantum algorithm that performs a quantum simulation of one dimensional quantum systems was presented in [99]. Figure 46 shows the following stages of the algorithm: (1) encoding the wave function on a N-point grid in a quantum state of $n = \log_2(N)$ qubits, $\psi \propto \psi(\hat{x})\,|q\rangle_{k_{\{n\}}}$ for $\hat{x} = \sum_{k=1}^{n} c_k \hat{Z}_k$ with $c_k = -2^{n-k-1}\Delta x$; (2) the Quantum Fourier Transform (QFT) to the momentum representation; (3) a diagonal phase transformation of the form $e^{-ik^2\Delta t}$; (4) the Inverse QFT back to the coordinate representation; (5) a diagonal phase transformation of the form $e^{-iV(x_i)\Delta t}$; (6) repetition of stages (2) through (5) in a loop until the desired time is reached. We will now consider a 2-qubit example of the quantum simulation algorithm in the case of a free particle, $V(x) = 0$.

Our initial wave function is a Π-function, which has $\{0, 1, 1, 0\}$ representation on a 2^n-point grid for $n = 2$ qubits. Its representation by the state of the qubits is $|0_{\hat{1}}, 1_{\hat{2}}\rangle + |1_{\hat{1}}, 0_{\hat{2}}\rangle$, which can be prepared up to a normalization constant on a quantum computer with $\hat{U}_{\text{prep}} = \hat{X}_{\hat{1}} \cdot C^{\hat{1}}\left[\hat{X}_{\hat{2}}\right] \cdot \hat{X}_{\hat{1}} \cdot \hat{H}_{\hat{1}}$ in terms of known gates.

We define the 2-qubit QFT as $\widehat{QFT} = \widehat{SWAP}_{\hat{1},\hat{2}}\hat{H}_{\hat{2}} \cdot C^{\hat{2}}\left[\hat{\mathcal{P}}_{\hat{1}}\left(\frac{\pi}{2}\right)\right] \cdot \hat{H}_{\hat{1}}$, where $\hat{\mathcal{P}}$ is a phase operator and $\widehat{SWAP}_{\hat{1},\hat{2}}$ is a $SWAP$ operation on the qubits. This transformation applies phase shifts to the probability amplitudes of the qubit state similar to the ones applied by the classical FFT to the function values. Hence, the resulting momentum representation is identical to the classical one in a sense that it is not centered around $k = 0$, which can be easily remedied by a single $\hat{X}_{\hat{1}}$ gate.

The momentum encoding adopted in this discussion is $k = -\frac{1}{2}\sqrt{\frac{\phi}{\Delta t}}\left(1 + \sum_{k=1}^{n} 2^{n-k}\hat{Z}_{\hat{k}}\right)$, where ϕ is a characteristic phase shift experienced by the state on a time step Δt. In this representation $-ik^2\Delta t$ phase shift contains one and two qubit contributions that commute with each other and could individually implemented. The one qubit phase shift gate has a straight forward implementation but the two qubit phase shift gate requires an ancillary qubit according to Ref. [75], which results in a three qubit implementation on a quantum computer. This implementation is captured in Figure 47 where removing the centering of the momentum representation and the inverse QFT have been added in order to return to the coordinate representation.

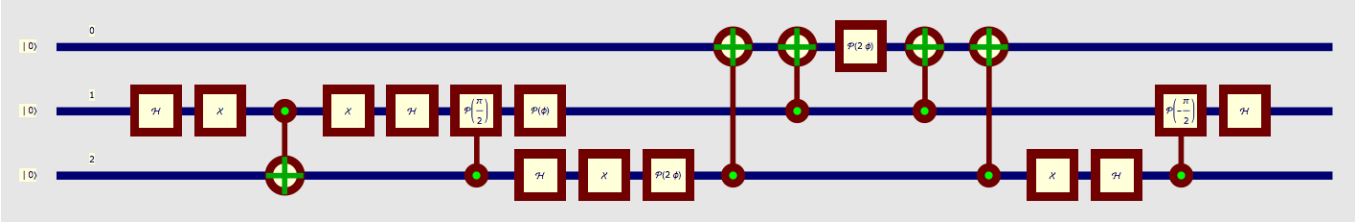


FIG. 47: The quantum circuit implementation of a 2-qubit algorithm that solves the Schrödinger's equation on a quantum computer. The initial state preparation is followed by the Quantum Fourier Transform and centering of the momentum representation. The single qubit phase shift transformations are followed by the two-qubit phase shift transformation that uses an ancillary qubit q[0]. The inverse Quantum Fourier Transform preceded by removing the centering operation completes the circuit and returns the wave function to the coordinate representation.

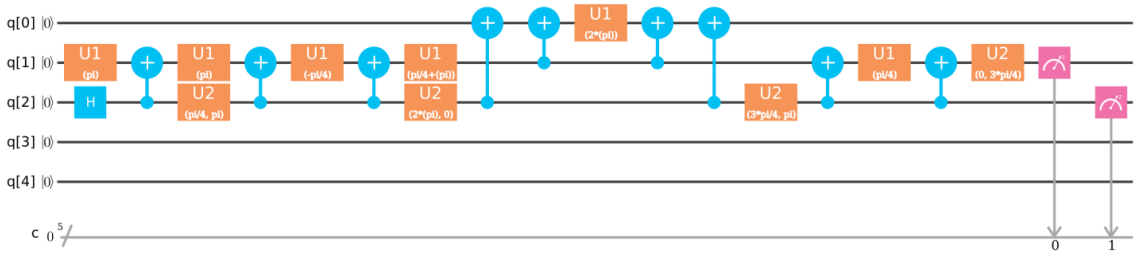


FIG. 48: The quantum circuit implementation of a 2-qubit algorithm that solves the Schrödinger's equation on the IBMQX4 quantum computer.

C. Algorithm implemented on IBM's 5-qubit computer

A real quantum computer implementation on the algorithm, see Figure 48, differs from the one presented in the previous section. This transition takes into account the topology of the chip and the availability of the gates such as $U1$ and $U2$. Finally, it performs a consolidation of the single qubit gates in order to reduce the number of physical operations on the qubits. The resulting QASM score is presented in the Listing 1.

```
include "qelib1.inc";
qreg q[3];
creg c[2];

u1(π)           q[1];
h               q[2];
cx          q[2], q[1];
u1(π)           q[1];
u2(π/4, π)      q[2];
cx          q[2], q[1];
u1(-π/4)        q[1];
cx          q[2], q[1];
u1(π/4+φ)       q[1];
u2(2φ, 0)       q[2];
cx          q[2], q[0];
cx          q[1], q[0];
u1(2φ)          q[0];
cx          q[1], q[0];
cx          q[2], q[0];
u2(3π/4, π)     q[2];
```

```
cx          q[2], q[1];
u1(π/4)           q[1];
cx          q[2], q[1];
u2(0, 3π/4)       q[1];

measure q[1] -> c[0];
measure q[2] -> c[1];
```

Listing 1: QASM score for a two qubit solution of the Schrödinger's equation in IBMQX4 topology: 1. an initial state of a Π-fuction preparation; 2. Quantum Fourier Transform; 3. Centering the momentum representation; 4. Quadratic phase shift $-ik^2\Delta t$; 5. Un-centering the momentum representation; 6. Inverse Quantum Fourier Transform; 7. Mesurements.

The QASM score of the Listing 1 has been tested on the IBMQX4 quantum chip, where the maximum number of executions in a single run is 2^{10}. The probabilities of observing qubit states in the computational basis have been measured for $\phi = 0$, $\phi = \pi/2$, $\phi = \pi$, $\phi = 3\pi/2$ and $\phi = 2\pi$. We expect that as ϕ increases from 0 to π the wave function evolves from a Π-function to a uniform function to a function peaked at the ends of the interval. The consecutive increase return the wave function back to the Π-function.

We have started with the $\phi = 0$ case that should have reproduced our initial state with ideal probabilities of {0, 0.5, 0.5, 0}. However, the observed probabilities were {0.173, 0.393, 0.351, 0.084}. Thus it was surprising to see that the $\phi = \pi/2$ case was very close to expected probability of 0.25 with the observed values of {0.295, 0.257, 0.232, 0.216}. This surprise was however short lived as the $\phi = \pi$ case has reverted back large errors for observed probabilities: {0.479, 0.078, 0.107, 0.335}. The final two case had the following observed probabilities {0.333, 0.248, 0.220, 0.199} and {0.163, 0.419, 0.350, 0.068} respectively.

D. Conclusion

In conclusion, we have outlined the necessary steps in order to implement the quantum simulation of the Schrödinger equation. These steps involve preparation of an initial state, Quantum Fourier Transform and diagonal phase transforms. Our 2-qubit example illustrates this simulations in the case of a free particle wave function a 4-point grid. This toy example has a straightforward translation to a real quantum chip implementation in the case of IBMQX4 topology.

We have tried to reduce the number of physical gate operations needed in order to implement this algorithm. However, simulations performed on a real chip revealed significant probability errors when detecting qubit states in the computational basis.

XVII. QUANTUM SIMULATION OF THE TRANSVERSE ISING MODEL

We present a quantum simulation of the transverse Ising model using the QISKIT simulator. The ground state of the transverse Ising model is calculated using the variational quantum eigenvalue solver, and is compared to the exact results.

A. Introduction

Novel properties in quantum materials arise from the strong quantum correlation between electrons. It is well-known that the dynamics of electrons in materials are described by the Schrödinger equation. Solving such a complex equation is a daunting task. To describe certain physical properties, one can resort to a minimal model by only keeping the essential interactions between electrons. One example is the Hubbard model, which is a canonical model in condensed matter physics (CMP) to describe the strong correlation between electrons. Over the past half century, many efforts have been made to solve the Hubbard model using classical computers. While significant progresss has been made, the model is still far from being completely solved. The difficulties lie in the quantum nature of electrons: Firstly, the Hilbert space required to describe the electronic wave function grows exponentially with the system size; secondly, there exists strong quantum entanglement between electrons even when they are spatially well separated; thirdly, the fermionic statistics of electrons does not have classical counterpart. All these quantum behaviors of electrons make the simulations of electrons using classical computers less efficient and difficult.

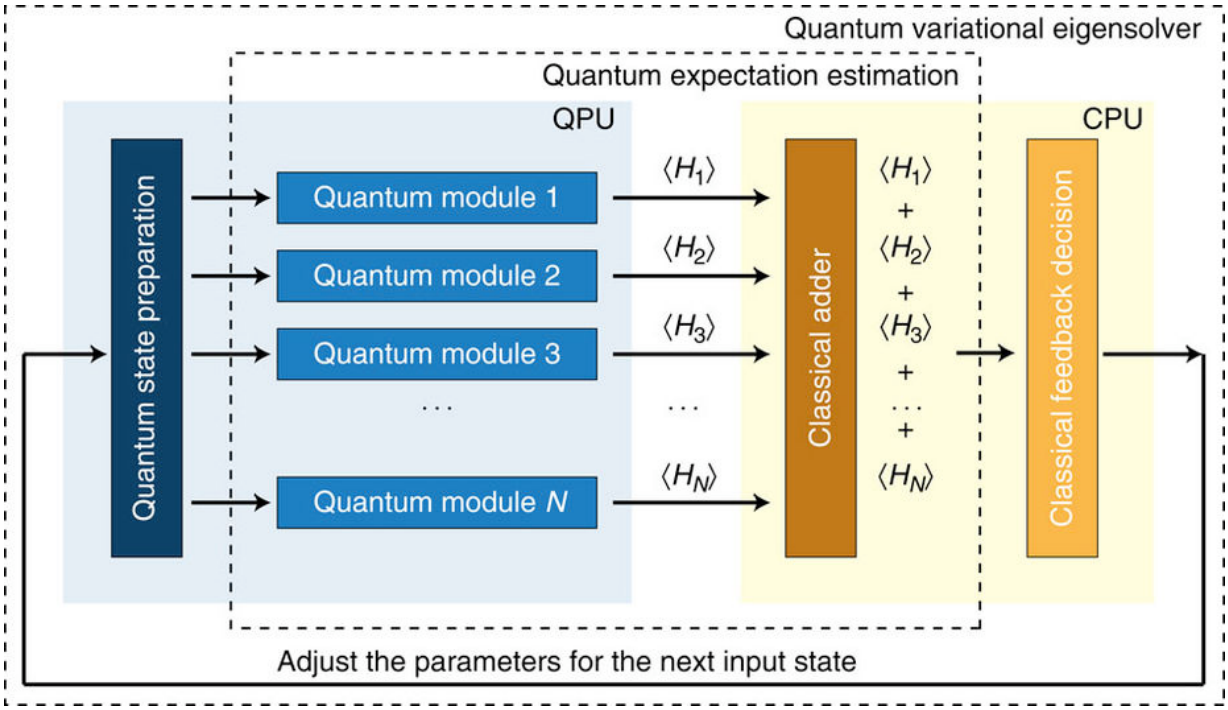


FIG. 49: Schematic view of the implementation of the variational quantum eigenvalue solver using a hybrid classical and quantum circuit. The figure is adopted from Ref. 80.

A quantum computer capable of calculating the evolution of a quantum state, on the other hand, could potentially become an efficient workhorse to simulate the many body electronic dynamics: a general-purpose quantum computers could solve quantum many-body problems that are not tractable using classical computers.

B. Variational quantum eigenvalue solver

A central task in CMP is finding the ground state of a given Hamiltonian, $\mathcal{H}$,

$$\mathcal{H}|\Psi\rangle = E_g|\Psi\rangle. \tag{80}$$

Once we know $|\Psi\rangle$, we can deduce the physical properties from the wave function. In this report, we will experiment with the IBM quantum simulator and the actual quantum device by finding the ground state energy of the transverse Ising model.

To find the eigenvalue of a Hamiltonian, we could use the quantum phase estimation algorithm, which utilizes the quantum interference between wave functions. The controlled unitary operator U is given by $\exp(-i\mathcal{H}\delta t/\hbar)$, where δt is the time step. By preparing different initial states $|\psi_i\rangle$ and repeat the measurement many times, in principle one can obtain the whole spectrum of the eigenvalues and the corresponding eigenwave functions. For a general Hamiltonian, however, the implementation of a controlled U may be not straightforward. For realistic problems, the quantum phase estimation requires large depth of the quantum circuit, which implies the need for long coherent time of the qubits, which is not available at the time of writing. For CMP problems, we are mainly interested in the lowest eigenvalue for most cases.

To overcome these limitations, we use the recently developed variational quantum eigenvalue solver (VQES) [72, 80]. The basic idea is to take the advantages of both the quantum and classical computers, as shown in Fig. 49. It allocates the task friendly to classical computers to classical computers and the other tasks to quantum computers. The algorithm is summarized as follows:

1. Prepare a variational state $|\psi(\theta_i)\rangle$ with parameters θ_i. For an efficient algorithm, the number of variational parameters should grow linearly with the system size.

2. Calculate the expectation value at quantum computers $E = \langle\psi|\mathcal{H}|\psi\rangle/\langle\psi|\psi\rangle$.

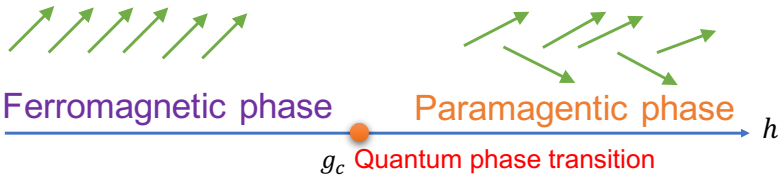


FIG. 50: Schematic view of the quantum phases described by the transverse Ising model. The arrows represent the spin configuration in the ordered and disordered phases.

3. Using classical nonlinear optimizer algorithms to find new optimal θ_i. In this report, we will use the relaxation method $\tau_0 \partial_t \theta_i = -\partial E/\partial \theta_i$, where τ_0 is a parameter to control the relaxation rate.

4. Iterate this procedure until convergence.

VQES has the following advantage: For most CMP problems, where the interaction is local, we can split the Hamiltonian into summation over many terms. This means that we can parallelize the algorithm to speed up the computation. The quantum expectation calculations for one term in the Hamiltonian are relative simple, thus no long coherence time for the quantum devices may be required. On the other hand, VQES also has limitations. Because of its variational nature, the trial wave function needs careful preparation. This requires physical insights into the problem under study. The ground state eigenvalue and eigenwave function are biased by the choice of the trial wave function. In addition, VQES requires communications between classical and quantum computers, which could be a bottleneck for the performance.

C. Simulation and results

We use VQES to find the ground state of the transverse Ising model (TIM) defined by

$$\mathcal{H} = -\sum_i \sigma_i^z \sigma_{i+1}^z - h \sum_i \sigma_i^x, \tag{81}$$

where σ^z, σ^x are Pauli matrices and h is the external magnetic field. Let us first review briefly the physical properties of this Hamiltonian. This Hamiltonian is invariant under the global rotation of spin along the x axis by π, $R_x \mathcal{H} R_x^\dagger = \mathcal{H}$, where $R_x(\pi)$ is the rotation operator

$$R_x \sigma^x R_x^\dagger = \sigma^x, \quad R_x \sigma^z R_x^\dagger = -\sigma^z. \tag{82}$$

The TIM has two phases: When the transverse field h is small, the spins are ordered ferromagnetically and the rotational symmetry associated with R_x is broken. In the ordered phase, the quantum expectation value $\langle \sigma^z \rangle \neq 0$. Upon increasing h, there is a quantum phase transition from the ordered phase to the disordered phase where $\langle \sigma^z \rangle = 0$, where the rotational symmetry is restored. The phase diagram is shown schematically in Fig. 50.

Guided by the knowledge of the phase diagram, first we propose a product state as a trial wave function. The wave function can be written as

$$|\psi_i(\theta_i)\rangle = \prod_i U(\theta_i)|0_i\rangle. \tag{83}$$

Here $U(\theta_i)$ is the unitary operation which describes the spin rotation along the y axis by an angle θ_i,

$$U(\theta_i) = \begin{pmatrix} \cos(\theta_i/2) & -\sin(\theta_i/2) \\ \sin(\theta_i/2) & \cos(\theta_i/2) \end{pmatrix}.$$

with θ_i being the variational parameters and we have used the Bloch sphere representation for a qubit state. For the TIM, we calculate the expectation value of

$$E_{J,i} = -\langle \psi | \sigma_i^z \sigma_{i+1}^z | \psi \rangle, \quad E_{Z,i} = -\langle \psi | \sigma_i^x | \psi \rangle \tag{84}$$

The quantum circuit to perform the preparation of the state and calculation of the expectation value is shown in Fig. 51. We have

$$E_{J,i} = -[P(q_i = 0) - P(q_i = 1)][P(q_{i+1} = 0) - P(q_{i+1} = 1)], \tag{85}$$

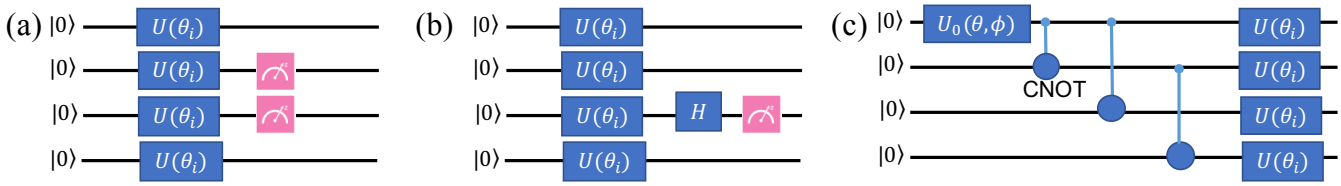


FIG. 51: Quantum circuits to perform the calculations described in the text.

$$E_{Z,i} = -[P(q_i = 0) - P(q_i = 1)], \tag{86}$$

where $P(q_i = 0, 1)$ is the measured probability for the qubit q_i in the $|0\rangle$ or $|1\rangle$ state. Because of the communication bottleneck, it is not possible to run the code using real quantum devices. We run the code using the quantum simulators through QISKIT package. The comparison of the results obtained from quantum simulation and analytical results are shown in Fig. 52. Our trial wave function works very well in the ordered phase, but the simulation results deviate from the exact solution in the quantum phase transition region. This discrepancy is caused by the fact that we have neglected the quantum entanglement in our trial wave function.

In a second set of experiments, we use a trial wave function that includes quantum entanglement. Because of symmetry, $|\Psi_i(\theta_i)\rangle$ and $R_x(\pi)|\Psi_i(\theta_i)\rangle$ are two degenerate wave functions with the same energy. The trial wave function can be written as a linear superposition of these two degenerate wave functions

$$|\psi_i(\theta_i)\rangle = \alpha|\Psi_i(\theta_i)\rangle + \beta R_x(\pi)|\Psi_i(\theta_i)\rangle. \tag{87}$$

The first step is to prepare $|\psi_i(\theta_i)\rangle$ using quantum circuit. To prepare an arbitrary state in a quantum circuit is not trivial as it requires of the order of 2^n CNOT gates, where n is the number of qubits [81]. The state in Eq. (87) can be prepared easily using the circuit in Fig. 51. Here we consider 4 spins. The first $U_0(\theta, \phi)$ operation transforms the state into

$$|0000\rangle \rightarrow e^{i\phi}\sin(\theta/2)|1000\rangle + \cos(\theta/2)|0000\rangle.$$

The first CNOT transforms the state into

$$e^{i\phi}\sin(\theta/2)|1100\rangle + \cos(\theta/2)|0000\rangle.$$

The second CNOT transforms the state into

$$e^{i\phi}\sin(\theta/2)|1110\rangle + \cos(\theta/2)|0000\rangle.$$

The third CNOT transforms the state into

$$e^{i\phi}\sin(\theta/2)|1111\rangle + \cos(\theta/2)|0000\rangle.$$

Finally we apply $U(\theta_i)$ rotation and we obtain the desired state in Eq. (87). Here

$$U_0(\theta, \phi) = \begin{pmatrix} \cos(\theta_i/2) & -\sin(\theta_i/2) \\ e^{i\phi}\sin(\theta_i/2) & e^{i\phi}\cos(\theta_i/2) \end{pmatrix}.$$

We then use VQES to find the ground state energy. As can be seen in Fig. 53, the new trial function almost reproduces the exact results in the whole magnetic field region, which improves the trial function using the product state.

D. Summary

We calculated the ground state of the transverse Ising model using the variational quantum eigenvalue solver implemented in the QISKIT simulator, and compared to the exact results. We have tried two variational trial wave

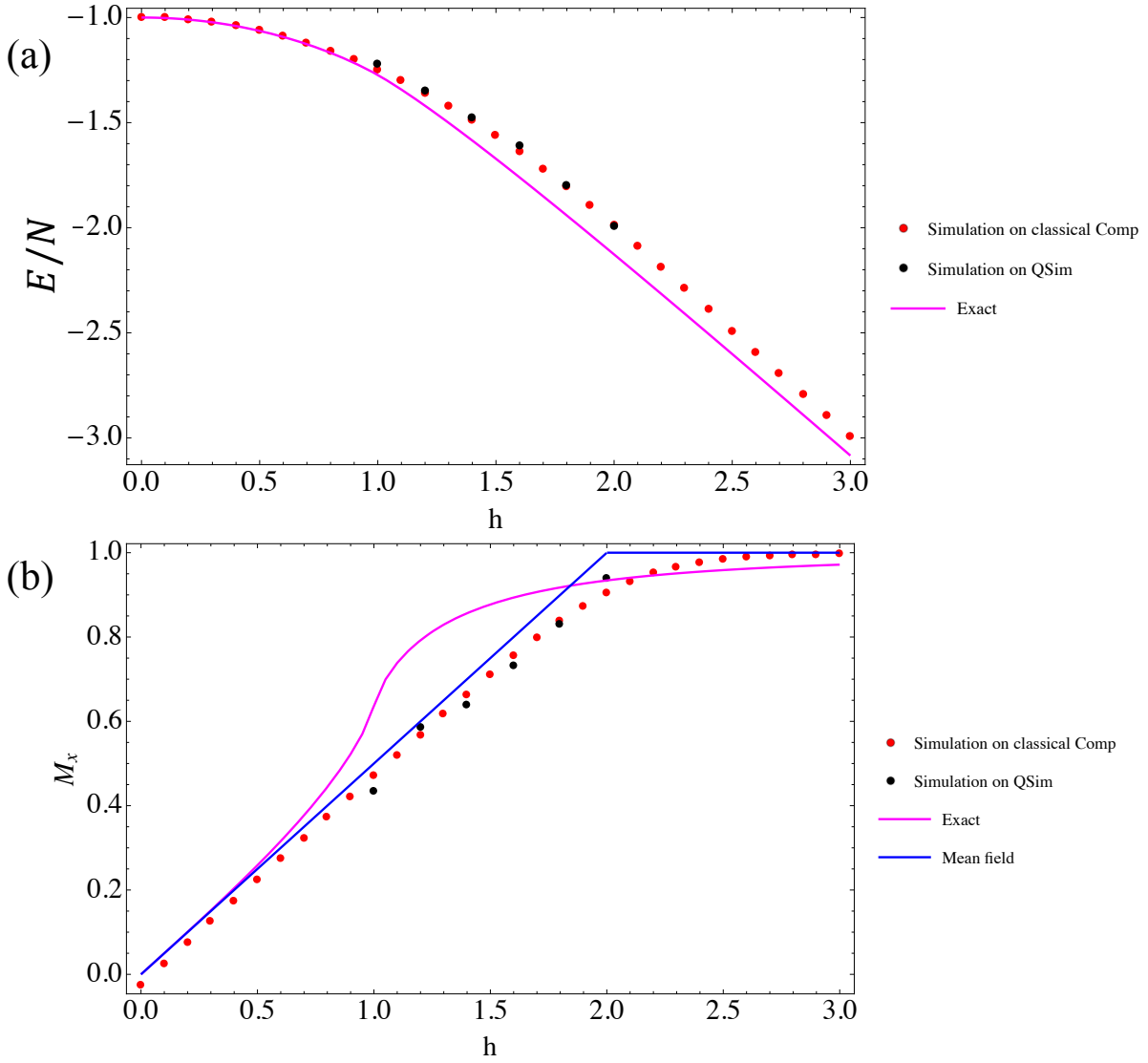


FIG. 52: Comparison of the ground state energy (a) and average magnetization (b) $M_x = \langle\psi|\sum_i \sigma_i^x|\psi\rangle/N$ obtained by using the trial wave functions in Eq. (83) and the exact results. Here we have used the periodic boundary condition. The simulations are run both on the quantum simulator (black symbols) and classical computers (red symbols). The mean-field results (blue line) are also displayed for comparison.

functions. For the product state trial wave function, it is in agreement with the exact results in the ordered phase. We then studied the trial wave function with entanglement between spins. The results obtained by using the new wave function are consistent with the exact results in the whole magnetic field region. Our case study demonstrates that the variational quantum eigenvalue solver can find the ground state efficiently with a carefully prepared trial wave function. However, when the quantum device and classical computers are not physically co-located do not locate in the same place, the algorithm suffers from a communication bottleneck.

XVIII. QUANTUM PARTITION FUNCTION

A. Background on the Partition Function

Calculation or approximation of the partition function is a sub-step of parameter estimation in Markov networks. The ability to learn the structure of a Markov network or to perform any inference on the network requires the calculation of the partition function [61]. Even for small networks, this calculation becomes intractable. Therefore, if

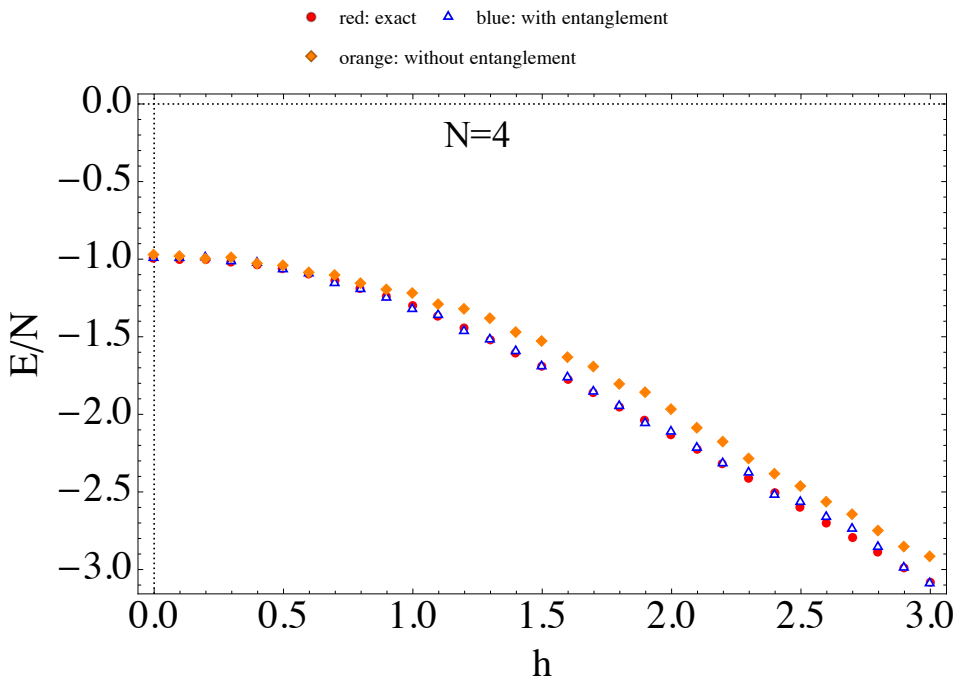


FIG. 53: (color online) Comparison of the ground state energy obtained by using the trial wave functions in Eqs. (83) and (87) and the exact result. Here we have used the periodic boundary condition. The number of spins is 4.

there exists an efficient quantum algorithm for the partition function, then many problems in graphical model inference and learning would become tractable and scalable; as well as other problems in computational physics [5, 42–44].

The partition function is of particular interest for calculating probabilities from graphical models such as Markov random fields [61]. For this article, we consider the graphical model form known as the Potts model. Let $\Gamma = (E, V)$ be a weighted graph with edge set E and vertex set V and $n = |V|$. In the q-state Potts model, each vertex can be in any of q discrete states. The Potts model is a generalization of the Ising model. In the Ising model $q = 2$; whereas in the Potts model $q \geq 2$. The edge connecting vertices i and j has a weight J_{ij} which is also known as the interaction strength between corresponding states. The Potts model Hamiltonian for a particular state configuration $\sigma = (\sigma_1, \ldots, \sigma_n)$ is

$$H(\sigma) = -\sum_{i \sim j} J_{ij} \delta_{\sigma_i, \sigma_j}, \tag{88}$$

where $i \sim j$ indicates that there exists an edge between vertices i and j; and where $\delta_{\sigma_i, \sigma_j} = 1$ if $\sigma_i = \sigma_j$ and 0 otherwise.

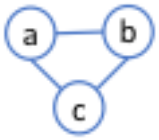
The probability of any particular configuration being realized in the Potts model at a given temperature, T, is given by the Gibbs distribution:

$$P(\sigma) = \frac{1}{Z(\beta)} e^{-\beta H(\sigma)} \tag{89}$$

where $\beta = 1/(k_B T)$ is the inverse temperature in energy units and k_B is the Boltzmann constant. The normalization factor, Z, is also known as the *partition function*:

$$Z(\beta) = \sum_{\{\sigma\}} e^{-\beta H(\sigma)} \tag{90}$$

where $\{\sigma\}$ means the full set of all possible state configurations. There are q^n possible state configurations, and so this is a sum over an infinite number of items and is generally intractable as well as difficult to approximate. The calculation of the partition function is #P-hard (i.e., it is a counting problem which is at least as hard as the NP-hard class of decision problems,). There is no known fully polynomial randomized approximation scheme (fpras), and it is unlikely that there is one [44].



(a) Graph with three vertices and three edges.

abc	$-H(\sigma)$
000	$J_{ab} + J_{bc} + J_{ac}$
001	J_{ab}
010	J_{ac}
011	J_{bc}
100	J_{bc}
101	J_{ac}
110	J_{ab}
111	$J_{ab} + J_{bc} + J_{ac}$

(b)

FIG. 54: (a) Simple example with (b) the enumeration of state configurations and the value of the Hamiltonian for a fully-connected 3-vertex Ising model ($q = 2$ Potts model)

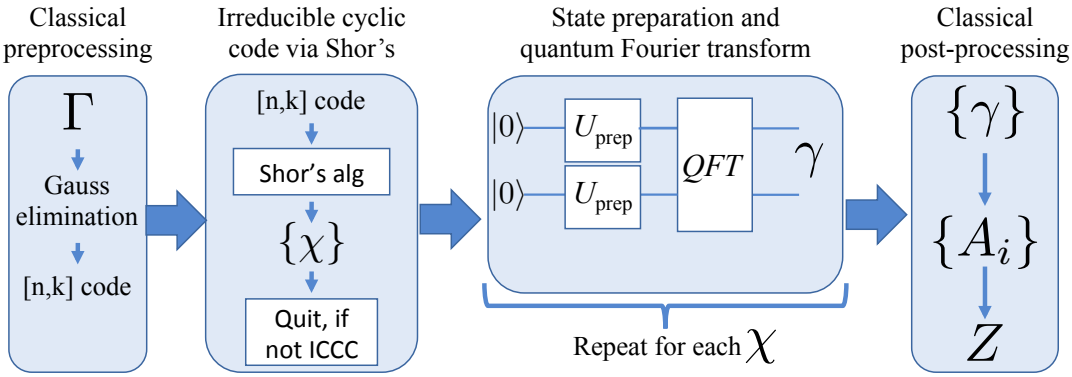


FIG. 55: Overview of the quantum partition function.

B. Simple example

We give a small example with a graph of $n = 3$, $V = \{a, b, c\}$, with edges between all pairs of vertices for three total edges, pictured in Figure 54a, and we use $q = 2$ for binary states on each vertex. To demonstrate the calculation of the partition function, we first enumerate the configurations as shown in Table 54b.

We plug the value of the Hamiltonian of each of the q^n configurations into the partition function given in Equation 90 to get the normalization constant:

$$Z(\beta) = 2e^{\beta(J_{ab}+J_{bc}+J_{ac})} + 2e^{\beta J_{ab}} + 2e^{\beta J_{bc}} + 2e^{\beta J_{ac}} \tag{91}$$

Letting $J_{ij} = 1$ for all $i \sim j$, gives:

$$Z(\beta) = 2e^{3\beta} + 6e^{\beta} \tag{92}$$

C. Calculating the Quantum Partition Function

An efficient quantum algorithm for the partition function is given by [44] for Potts models whose graph, Γ, has a topology such that it can be represented with an irreducible cyclic cocycle code (ICCC). This stipulation is non-intuitive and it takes a quantum algorithm to efficiently determine if a given graph meets this requirement. From the graph, Γ, calculate a cyclic code $C(\Gamma)$ that represents the generating structure of the graph by using Gaussian elimination on the incidence matrix of the graph, and then use Shor's algorithm to determine the irreducible set of code words, χ. If the code is not irreducible, then we will not be able to efficiently calculate the partition function for this graph.

Assuming that the given graph is ICCC, the first step in the partition function algorithm is to calculate the Gauss sum of $G_{\mathbf{F}_{q^k}} = \sqrt{q^k}e^{i\gamma}$, where γ is a function of χ. The difficult part is to calculate γ, which can be done efficiently using the quantum Fourier transform (QFT). Using the set of values, $\{\gamma\}$ for all of the words, $\{\chi\}$ in the code; we calculate the weight spectrum $\{A_i\}$ of the code representing Γ. From this weights spectrum, the partition function Z can be efficiently calculated using classical computing.

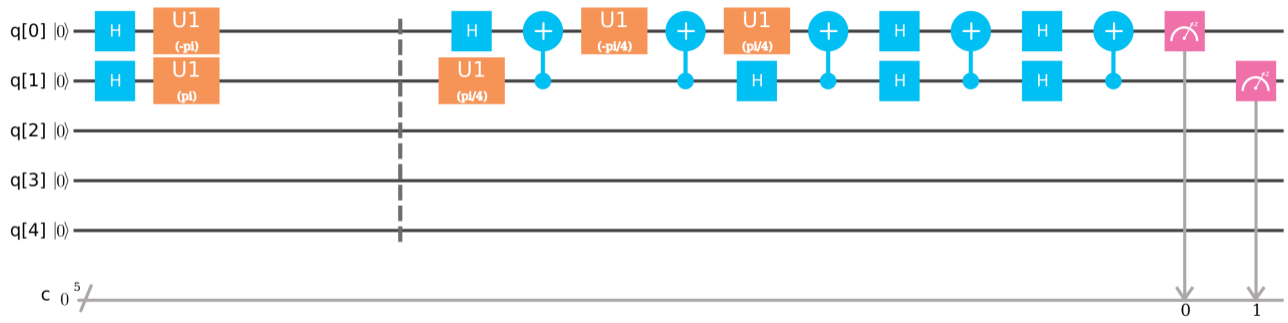


FIG. 56: Score of encoding and quantum Fourier transform on 2 qubits.

D. Implementation of a Quantum Algorithm on the IBM Quantum Experience

We implemented one step of the full partition function algorithm using the IBM Quantum Experience. The implemented algorithm is the 2-qubit quantum Fourier transform (QFT2), as the first step in actual calculation of the partition function. The input to this step is the irreducible cocyclic code. The irreducible cyclic code for the example problem of a 3-vertex Ising model is $[1, -1]$ with $n = |V| = 3$ and $k = |E| - c(\Gamma) = 2$, where $c(\Gamma)$ is the number of connected components in the graph Γ. This small example does meet the ICCC requirement (as checked through classical calculation), so we will continue with the calculation of the partition function of the example without implementing the quantum algorithm for checking the requirement. In the fully-connected 3-vertex Ising model example given, the input to QFT2 is $q[0] = |+\rangle$ and $q[1] = |-\rangle$. In the sample score shown in Fig 56, the encoding stage happens before the barrier. The QFT2 algorithm, as given by the Qiskit Tutorial provided by IBM, is the rest of the code. The output bits should be read in reverse order. Some gates could be added at the end of the QFT2 algorithm to read the gates in the same order as the input.

E. Results

The result from simulating 1000 shots gives $P(\gamma = 1) = 0.47$ and $P(\gamma = 3) = 0.53$. The result from running on the actual hardware is $P(\gamma = 0) = 0.077$, $P(\gamma = 1) = 0.462$, $P(\gamma = 2) = 0.075$, and $P(\gamma = 3) = 0.386$. We see that the hardware sometimes allows observations of $\gamma = 0$ and $\gamma = 2$, which should not be possible. We leave additional investigations to evaluate how badly this inaccuracy affects affect the overall results in calculating the partition function for future work.

XIX. QUANTUM STATE PREPARATION

The problem of preparing an n-qubit state consists first of finding the unitary transformation that takes the N-dimensional vector (1,0,...0) to the desired state $(\alpha_1, \ldots, \alpha_N)$, where $N = 2^n$, and then rendering the unitary transformation into a sequence of gates.

A. Single Qubit State Preparation

A single qubit quantum state $|\psi\rangle$ is represented as a superposition of “up" and “down" states $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, where $|\alpha|^2 + |\beta|^2 = 1$. The sizes $|\alpha|^2$ and $|\beta|^2$ represent the relative probability of $|\psi\rangle$ being up or down. Up to a non-observable global phase, we may assume that α is real, so that $|\psi\rangle = \cos\theta\,|0\rangle + e^{i\phi}\sin\theta\,|1\rangle$ for some angles θ, ϕ. In this way, we can represent the state as a point on the unit sphere with θ the co-latitude and ϕ the longitude. This is the well-known Bloch sphere representation. In this way, the problem of 1-qubit state preparation consists simply of finding the unitary transformation that takes the North pole to (α, β). In practice, this amounts to finding a sequence

of available gates on actual hardware that will leave the qubit in the desired state, to a specified desired accuracy. On the IBM platform, these gates are X, Y, Z, T, S, and H and their definitions are found in many standard references.

To prepare a specified state $|\psi\rangle$, we must find a 2×2 unitary matrix U taking the vector $|0\rangle$ to $|\psi\rangle$. An obvious simple choice for U is

$$U = \begin{pmatrix} \cos\theta & -\sin\theta e^{-i\phi} \\ \sin\theta e^{i\phi} & \cos\theta \end{pmatrix}$$

However, if our goal is to initialize a base state with the fewest possible standard gates, this may not be the best choice. Instead, it makes sense to consider a more general possible unitary operator whose first column is our desired base state, and then determine the requisite number of standard gates to obtain it.

Any 2×2 unitary matrix may be obtained by means of a product of three rotation matrices, up to a global phase

$$U = e^{i\alpha} R_z(\beta) R_y(\gamma) R_z(\delta)$$

where here $R_z(\beta) = \text{diag}(e^{i\beta/2}, e^{-i\beta/2})$ and $R_y(\gamma)$ is related to $R_z(\gamma)$ by $R_y(\gamma) = SHR_z(\gamma)HSZ$. The rotation matrices $R_y(\gamma)$ and $R_z(\beta)$ correspond to the associated rotations of the unit sphere under the Bloch representation. In this way, the above decomposition is a reiteration of the standard Euler angle decomposition of elements of $SO(3)$. Thus the problem of approximating an arbitrary quantum state is reduced to the problem of finding good approximations of $R_z(\gamma)$ for various values of γ.

There has been a great deal of work done on finding efficient algorithms for approximating elements $R_z(\gamma)$ using universal gates to a specified accuracy. However, these algorithms tend to focus on the asymptotic efficiency: specifying approximations with the desired accuracy which are the generically optimal in the limit of small errors. From a practical point of view, this is an issue on current hardware, since representations tend to involve hundreds of standard gates, far outside the realm of what may be considered practical For this reason, it makes sense to ask the question of how accurately one may initialize an arbitrary qubit with a specified number of gates.

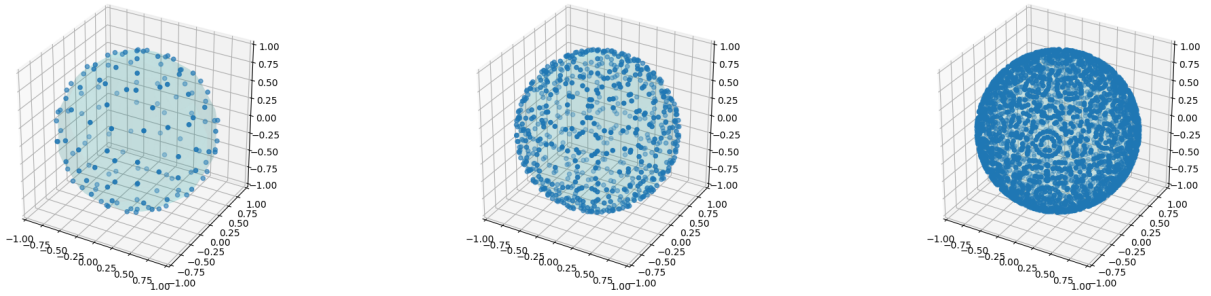


FIG. 57: Possible exact state initializations using $10, 15$, and 20 gates. With 20 gates, every point on the sphere is within approximately 0.072 of an exactly obtainable state. With 30 gates, every point is within 0.024

We empirically observe that the maximum possible chordal distance from a point on the Bloch sphere to the set of exact states decreases exponentially with the number of gates. With 30 gates, every point is within 0.024 of a desired gate. Thus, to within an accuracy of about 2.5%, we can represent any base state as a product of about 30 states. We do so by preserving the states generated by 30 gates, and then for any point finding the closest exact point.

On examining the generated QASM code, however, we found out that the IBM has the U gate implemented in the hardware

B. Schmidt Decomposition

The initialization of qubit states using more than one qubit is aided by the so-called Schmidt decomposition, which we now introduce. Specifically, the Schmidt decomposition allows one to initialize a $2n$-qubit state by initializing a single n-qubit state, along with two specific n-qubit gates, combined together with n CNOT gates.

Mathematically, an arbitrary $2n$-qubit state $|\psi\rangle$ may be represented as a superposition

$$|\psi\rangle = \sum_{i_1,\ldots,i_n \in \{0,1\}} \sum_{j_1,\ldots,j_n \in \{0,1\}} a_{i_1,\ldots,i_n,j_1,\ldots,j_n} \, |i_1 i_2 \ldots i_n j_1 j_2 \ldots j_n\rangle \, .$$

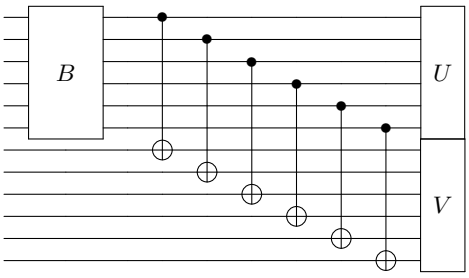


FIG. 58: Schmidt decomposition

In a Schmidt decomposition, we obtain such a state by strategically choosing two orthonormal bases $|\xi_j\rangle, |\varphi_j\rangle$ for $j = 1, \ldots, 2^n$ of the Hilbert space of n-qubit states and then writing $|\psi\rangle$ as the product

$$|\psi\rangle = \sum_{i=1}^{2^n} \lambda_i |\xi_i\rangle |\varphi_i\rangle,$$

for some well-chosen λ_i's.

The bases $|\xi_j\rangle$ and $|\varphi_j\rangle$ may be represented in terms of two unitary matrices $U, V \in U(2^n)$, while the λ_i's may be represented in terms of a single n-qubit state. We represent this latter state as $B|00\ldots0\rangle$ for some $B \in U(2^n)$. Then from a quantum computing perspective, the product in the Schmidt decomposition may be accomplished by a quantum circuit combining U, V, and B with n CNOT gates as shown below for $n = 6$.

Let C_i^j denote the CNOT operator with control j and target i. Algebraically, the above circuit may be written as a unitary operator $T \in U(2^{2n})$ of the form

$$T = (U \otimes V)(C_{n+1}^1 \otimes C_{n+2}^2 \otimes \cdots \otimes C_{2n}^n)(B \otimes I).$$

We will use $|e_1\rangle, \ldots, |e_{2^n}\rangle$ to denote the standard basis for the space of n-qubit states, in the usual order. We view each of the elements e_j as a vector in $\{0,1\}^n$. In this notation, the formation of CNOT gates above acts on simple tensors by sending

$$C_{n+1}^1 \otimes C_{n+2}^2 \otimes \cdots \otimes C_{2n}^n : |e_i\rangle |e_j\rangle \mapsto |e_i\rangle |e_i + e_j\rangle, \quad e_i, e_j \in \{0,1\}^n,$$

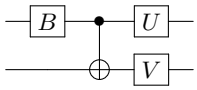
where addition in the above is performed modulo 2. Therefore the action of the operator T associated to the above circuit on the basis vector $|00\ldots0\rangle$ is

$$\begin{aligned}
T|00\ldots0\rangle &= (U \otimes V)(C_{n+1}^1 \otimes C_{n+2}^2 \otimes \cdots \otimes C_{2n}^n)(B \otimes I)|00\ldots0\rangle \\
&= (U \otimes V)(C_{n+1}^1 \otimes C_{n+2}^2 \otimes \cdots \otimes C_{2n}^n) \sum_{i=1}^{2^n} b_{i1} |e_i\rangle |e_1\rangle \\
&= (U \otimes V) \sum_{i=1}^{2^n} b_{i1} |e_i\rangle |e_i\rangle \\
&= \sum_{i=1}^{2^n} b_{i1} (U|e_i\rangle)(V|e_i\rangle) = |\psi\rangle
\end{aligned}$$

Thus we see that the above circuit performs precisely the sum desired from the Schmidt decomposition.

To get the precise values of U, V, and B, we write $|\psi\rangle = \sum_{i,j=1}^{2^n} a_{ij} |e_i\rangle |e_j\rangle$ for some constants $a_{ij} \in \mathbb{C}$ and define A to be the $2^n \times 2^n$ matrix whose entries are the a_{ij}'s. Then comparing this to our previous expression for $|\psi\rangle$, we see

$$\sum_{i,j=1}^{2^n} a_{ij} |e_i\rangle |e_j\rangle = \sum_{k=1}^{2^n} b_{k1} (U|e_k\rangle)(V|e_k\rangle).$$



The choiceof U, V, and B are covered comprehensively in the Schmidt decomposition description above.

FIG. 59: Circuit for two qubit-state preparation

Multiplying on the left by $\langle e_i| \langle e_j|$ this tells us

$$a_{ij} = \sum_{k=1}^{2^n} b_{k1} u_{ik} v_{jk},$$

where here $u_{ik} = \langle e_i| U |e_k\rangle$ and $v_{jk} = \langle e_j| V |e_k\rangle$ are the i,k'th and j,k'th entries of U and V, respectively. Encoding this in matrix form, this tells us

$$V \mathrm{diag}(b_{i1}, \ldots, b_{in}) U^T = A.$$

Then to calculate the value of U, V and the b_{i1}'s, we use the fact that V is unitary to calculate:

$$A^\dagger A = U^{T\dagger} \mathrm{diag}(|b_{i1}|^2, \ldots, |b_{in}|^2) U^T.$$

Thus if we let $|\lambda_1|^2, \ldots, |\lambda_n|^2$ be the eigenvalues of $A^\dagger A$, and let U to be a unitary matrix satisfying

$$U^T A^\dagger A U^{T\dagger} = \mathrm{diag}(|\lambda_1|^2, \ldots, |\lambda_N|^2),$$

let $b_{i1} = \lambda_i$ for $i = 1, \ldots, n$ and let

$$V = A U^{T\dagger} \mathrm{diag}(\lambda_1, \ldots, \lambda_n)^{-1}.$$

The matrix U is unitary, and one easily checks that V is therefore also unitary. Moreover $\sum_i |b_{i1}|^2 = \mathrm{Tr}(A^\dagger A) = \sum_i |a_{ij}|^2 = 1$, and so the b_{i1}'s are representative of an n-qubit state and can be taken as the first column of B.

C. Two-qubit State Preparation

An arbitrary two-qubit state $|\psi\rangle$ is a linear combination of the four base states $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ such that the square sum of the magnitues of the coefficients is 1. In terms of a quantum circuit, this is the simplest case of the circuit defined above in the Schmidt decomposition, and may be accomplished with three 1-qubit gates and exactly 1 CNOT gate, as featured below.

D. Two-qubit Gate (Unitary Operator) Preparation

In order to initialize a four-qubit state, we require the initialization of arbitrary two-qubit gates. A two-qubit gate may be represented as an element U of $SU(4)$. As it happens, any element of $U(4)$ may be obtained by means of precisely 3 CNOT gates, combined with 7 1-qubit gates arranged in a circuit of the form The proof of this is nontrivial and relies on a characterization of the image of $SU(2)^{\otimes 2}$ in $SU(4)$ using the Makhlin invariants. We do not aim to reproduce the proof here. Instead, we merely aim to provide a recipe by which one may successfully obtain any element of $SU(4)$ via the above circuit and an appropriate choice of the one-qubit gates.

Let $U \in SU(4)$ be the element we wish to obtain. To choose A, B, C, D and the R_i's, let C_j^i denote the CNOT gate with control on qubit i and target qubit j and define α, β, δ by

$$\alpha = \frac{x+y}{2},\ \beta = \frac{x+z}{2},\ \delta = \frac{y+z}{2}$$

for e^{ix}, e^{iy}, e^{iz} the eigenvalues of the operator $U(Y \otimes Y)U^T(Y \otimes Y)$. Then set

$$R_1 = R_z(\delta), R_2 = R_y(\beta), R_3 = R_y(\alpha), E = C_1^2(S_z \otimes S_x)$$

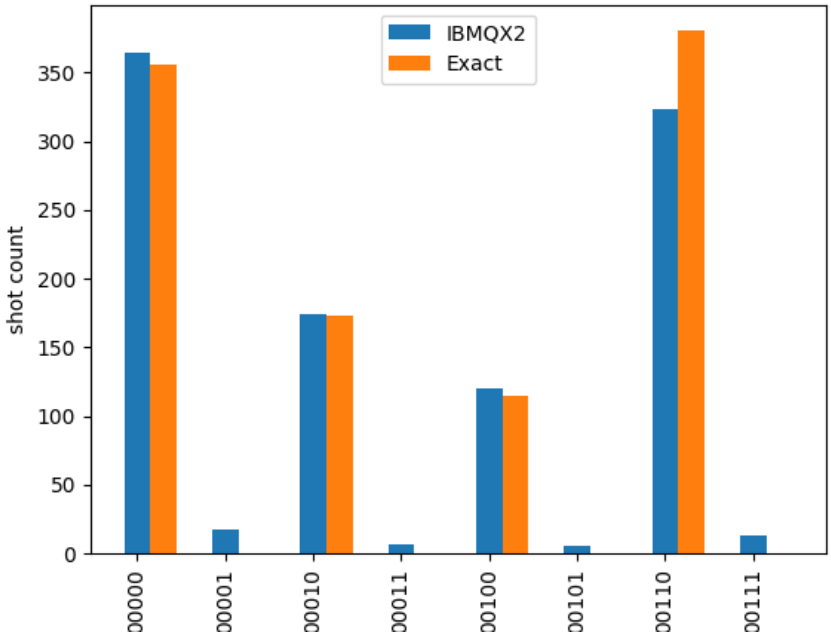


FIG. 60: Verification on IBMQX2

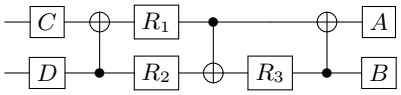


FIG. 61: Circuit implementation of an arbitrary two qubit gate.

and also

$$V = e^{i\pi/4}(Z \otimes I)C_1^2(I \otimes R_3)C_2^1(R_1 \otimes R_2)C_1^2(I \otimes S_z^\dagger).$$

Define $\widetilde{U}, \widetilde{V}$ by $\widetilde{U} = E^\dagger U E$ and $\widetilde{V} = E^\dagger V E$. Let $\widetilde{A}, \widetilde{B}$ be the real, unitary matrices diagonalizing the eigenvectors of $\widetilde{U}\widetilde{U}^T$ and $V\widetilde{V}^T$, respectively. Set $X = \widetilde{A}^T\widetilde{B}$ and $Y = V^\dagger\widetilde{B}^T\widetilde{A}U$. Then $EXE^\dagger$ and $EYE^\dagger$ are in $SU(2)^{\otimes 2}$ and we choose A, B, C, D such that

$$(AS_Z^\dagger) \otimes (Be^{i\pi/4}) = EXE^\dagger \quad \text{and} \quad C \otimes (S_z D) = EYE^\dagger.$$

By virtue of this construction, the above circuit is algebraically identical to U.

E. Four Qubit State Preparation

From the above results that any two-qubit state requires 1 CNOT gate, any two-qubit operator requires three CNOT gates, and the Schmidt decomposition, we see that we should be able to write a circuit initializating any four-qubit state with only 9 CNOT gates in total, along with 17 one-qubit gates. This represents the second most simple case of the Schmidt decomposition, which we write in combinantion with our generic expression for 2-qubit gates as The above circuit naturally breaks down into four distinct stages, as shown by the separate groups surrounded by dashed lines. During the first stage, we initialize the first two qubits to a specific state relating to a Schmidt decomposition of the full 4 qubit state. Stage two consists of two CNOT gates relating the first and last qubits. Stages three and four are generic circuits representing the unitary operators associated to the orthonormal bases in the Schmidt decomposition.

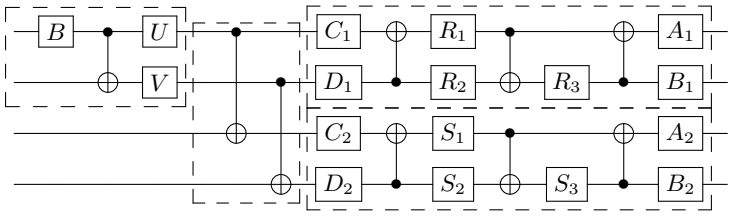


FIG. 62: Circuit for four qubit-state preparation. The four phases of the circuit are indicated in dashed boxes.

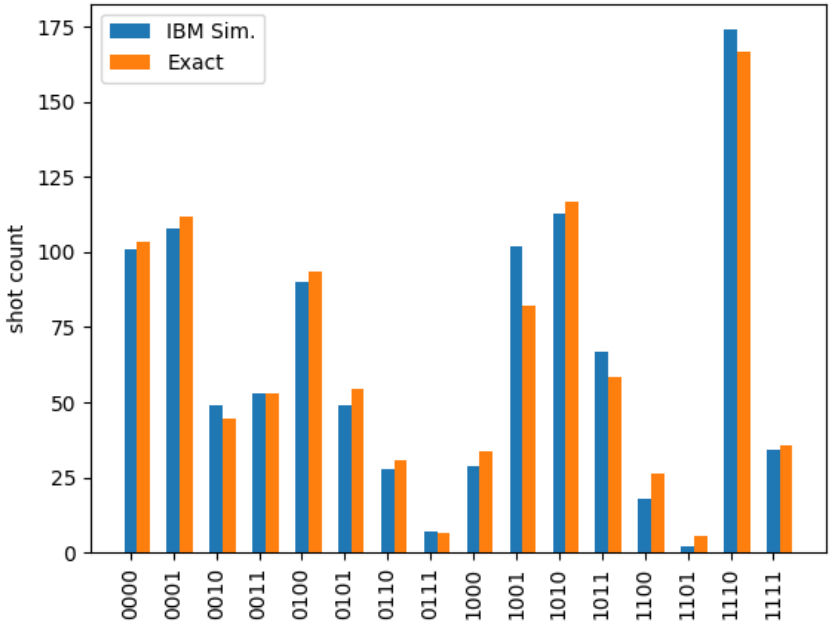


FIG. 63: Verification of the quantum circuit for four qubit-state preparation

XX. QUANTUM TOMOGRAPHY

A. Problem definition and background

Quantum state estimation, or tomography, deals with the reconstruction of the state of a quantum system from measurements of several preparations of this state. In the context of quantum computing, a two-qubit example is presented in Figure 64. Imagine that we start with the state $|00\rangle$, and apply some quantum algorithm (represented by a unitary matrix U) to the initial state, thus obtaining a state $|\psi\rangle$. We can measure this state in the computational z basis, or apply some rotation (represented by V) in order to perform measurements in a different basis. Quantum tomography aims to answer the following question: is it possible to reconstruct the state $|\psi\rangle$ from a certain number of such measurements? Hence, quantum tomography is not a quantum algorithm *per se*, but it is an important procedure for certifying the performance of quantum algorithms and assessing the quality of the results that can be corrupted by decoherence, environmental noise, and biases, inevitably present in analogue machines, *etc.* Moreover, as illustrated in Figure 64, similar procedures can be used for certifying the initial state, as well as for measuring the fidelity of gates.

A unique identification of state requires a sufficient number of *tomographically complete* measurements, meaning that the algorithm should be run several times. Unfortunately, because of the noise, it is impossible to obtain the exact same state $|\psi\rangle$ every time; instead, one should see a mixture of different states: $|\psi_1\rangle$, $|\psi_2\rangle$, ..., $|\psi_k\rangle$. In general, there does not exist a single $|\psi\rangle$ describing this mixture. Therefore, we need a different definition of the state.

Let us denote p_i the probability of occurrence of the state $|\psi_i\rangle$. We define the *density matrix* as

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|. \tag{93}$$

Using this more general definition of the state, the expected value of an observable A is given by $\langle A\rangle = \sum_i p_i \mathrm{Tr}\langle\psi_i|A|\psi_i\rangle = \mathrm{Tr}(A\rho)$. The density matrix has the following properties:

- $\mathrm{Tr}\,\rho = 1$, i.e., probabilities sum to one
- $\rho = \rho^\dagger$, and $\rho \succcurlyeq 0$, i.e., all eigenvalues are positive or zero

In a popular setting for quantum tomography [75], the set of measurement operators P_i are taken as projectors that form several *Postive Operator-Valued Measures* (POVM), i.e., they satisfy $\sum_i P_i = I$. For single qubits, examples of such projectors in the computational basis are given by $P_0 = |0\rangle\langle 0|$ and $P_1 = |1\rangle\langle 1|$, and in the x-basis by $P_\pm = \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle) \otimes \frac{1}{\sqrt{2}}(\langle 0| \pm \langle 1|)$. Assume that the set of projectors that we take represents a *quorum*, i.e., it provides sufficient information to identify the state of the system in the limit of a large number of observations, and that for each subset forming a POVM m measurements are collected. Given the occurencies m_i for each projector P_i, we can define the associated empirical frequency as $\omega_i = m_i/m$. Then the quantum tomography problem can be stated as follows: reconstruct ρ from the set of couples of projectors and measurement frequencies $\{P_i, \omega_i\}$. In other words, we would like to "match" $\mathrm{Tr}(P_i\rho)$ and ω_i. The next section presents a short overview of most popular general methods for the quantum state estimation.

B. Short survey of existing methods

Most popular methods for quantum tomography in the general case include:

1. **Linear inversion.** In this method, we simply aim at inverting the system of equations $\mathrm{Tr}(P_i\rho) = \omega_i$. Although being fast, for a finite number of measurements thus obtained estimation $\widehat{\rho}$ does not necessarily satisfy $\widehat{\rho} \succcurlyeq 0$ (i.e., might contain negative eigenvalues) [55].

2. **Linear regression.** This method corrects for the disadvantages of the linear inversion by solving a constrained quadratic optimization problem [83]:

$$\widehat{\rho} = \underset{\rho}{\mathrm{argmin}} \sum_i [\mathrm{Tr}(P_i\rho) - \omega_i]^2 \quad \text{s.t. } \mathrm{Tr}\,\rho = 1 \text{ and } \rho \succcurlyeq 0$$

   The advantage of this method is that data does not need to be stored, but only the current estimation can be updated in the streaming fashion. However, this objective function implicitly assumes that the residuals are Gaussian-distributed, which does not necessarily hold in practice for a finite number of measurements.

3. **Maximum likelihood.** In this by far most popular algorithm for quantum state estimation, one aims at maximizing the log-probability of observations [50, 55]

$$\widehat{\rho} = \underset{\rho}{\mathrm{argmax}} \sum_i \omega_i \ln \mathrm{Tr}(P_i\rho) \quad \text{s.t. } \mathrm{Tr}\,\rho = 1 \text{ and } \rho \succcurlyeq 0$$

   This is a convex problem that outputs a positive semidefinite (PSD) solution $\widehat{\rho} \succcurlyeq 0$. However, it is often stated that the maximum likelihood (ML) method is slow, and several recent papers attempted to develop faster methods of gradient descent with projection to the space of PSD matrices, see e.g. [91]. Among other common criticisms of this method one can name the fact that ML might yield rank-deficient solutions, which results in an infinite conditional entropy that is often used as a metric of success of the reconstruction.

4. **Bayesian methods.** This is a slightly more general approach compared to the ML method which includes some prior [12], or corrections to the basic ML objective, see e.g., the so-called Hedged ML [11]. However, it is not always clear how to choose these priors in practice. Markov Chain Monte Carlo Methods that are used for general priors are known to be slow.

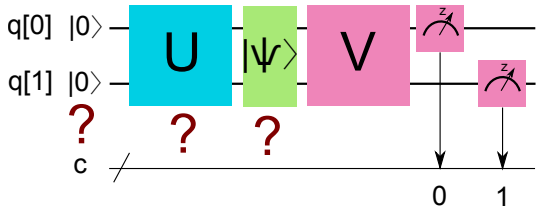


FIG. 64: Quantum tomography in the context of quantum computing: using a certain number of measurements in different basis implemented with V, is it possible to reconstruct the system state $|\psi\rangle$ after application of the quantum algorithm U, as well as certify the initial state and assess the fidelity of the gates in U?

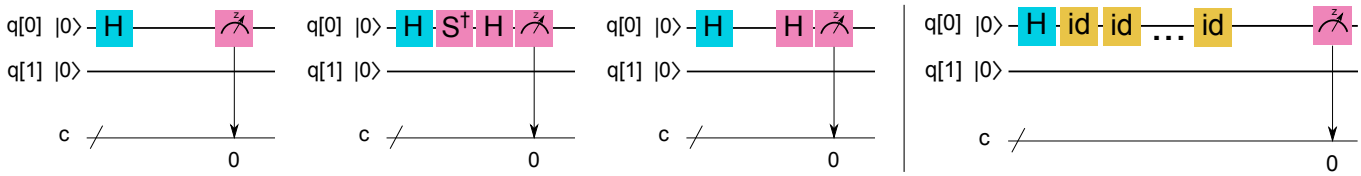


FIG. 65: Left: measurements of the single qubit state after the application of the Hadamard gate, in z, y and x basis. Right: experimental setup for testing the effects of decoherence.

Let us mention that there exist other state reconstruction methods that attempt to explore a particular known structure of the density matrix, such as compressed-sensing methods [47] in the case of low-rank solutions, and matrix product states [28] or neural networks based approaches [105] for pure states with a limited number of entanglements, *etc.* One of the points we can conclude from this section is that the ultimately best general method for the quantum state tomography is not yet known. However, it seems that maximum likelihood is still the most widely discussed method in the literature; in what follows, we implement and test ML approach to quantum tomography on the IBM quantum computer.

C. Implementation of Maximum Likelihood method on 5-qubit IBM QX

We present an efficient implementation of the ML method using a fast gradient descent with an optimal 2-norm projection [98] to the space of PSD matrices[2]. In what follows, we apply quantum tomography to study the performance of the IBM QX.

1. *Warm-up: Hadamard gate*

Let us start with a simple one-qubit case of the Hadamard gate, see Figure 65. This gate transforms the initial qubit state $|0\rangle$ as follows: $H:\ |0\rangle \to |+\rangle_x = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, so that the density matrix should be close to $\rho = |+\rangle_x\langle +|_x$. In the limit of a large number of measurements, we expect to see the following frequencies in the z, y, and x basis (all vector expressions are given in the computational basis):

$$\begin{bmatrix}1\\0\end{bmatrix} \to \frac{1}{2}\,, \quad \begin{bmatrix}0\\1\end{bmatrix} \to \frac{1}{2}\,, \qquad \frac{1}{\sqrt{2}}\begin{bmatrix}1\\i\end{bmatrix} \to \frac{1}{2}\,, \quad \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-i\end{bmatrix} \to \frac{1}{2}\,, \qquad \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} \to 1\,, \quad \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix} \to 0\,.$$

We learn the estimated density matrix $\widehat{\rho}$ from measurements in each basis using the maximum likelihood method, and look at the decomposition:

$$\widehat{\rho} = \lambda_1 |\psi_1\rangle\langle\psi_1| + \lambda_2 |\psi_2\rangle\langle\psi_2|$$

which would allow us to see what eigenstates contribute to the density matrix, and what is their weight. Indeed, in the case of ideal observations we should get $\lambda_1 = 1$, with $|\psi_1\rangle = \begin{bmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\end{bmatrix}^T$, and $\lambda_2 = 0$ with $|\psi_2\rangle = \begin{bmatrix}\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{bmatrix}^T$, corresponding to the original pure state associated with $|+\rangle_x$.

---

[2] The julia implementation of the algorithm is available at `http://gitlab.lanl.gov/QuantumProgramming2017/QuantumTomography`

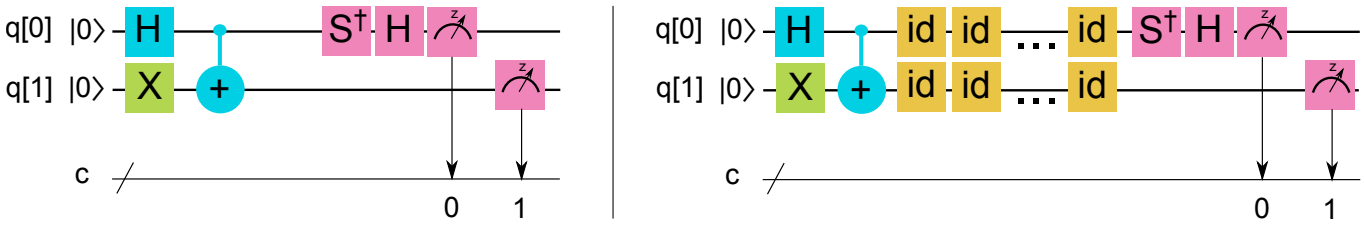


FIG. 66: Left: example of a measurement of the two-qubit maximally entangled state created with the combination of H, X and $CNOT$ gates in the yz basis. Right: experimental setup for testing the effects of decoherence.

Instead, we obain the following results for the eigenvalues and associated eigenvectors after 8152 measurements (the maximum number in one run on IBM QX) in each basis (z, y, x):

$$\lambda_1 = 0.968 \rightarrow \begin{bmatrix} 0.715 - 0.012i \\ 0.699 \end{bmatrix} \qquad \lambda_2 = 0.032 \rightarrow \begin{bmatrix} 0.699 - 0.012i \\ -0.715 \end{bmatrix},$$

i.e., in 96% of cases we observe the state close to $|+\rangle_x$, and the rest corresponds to the state which is close to $|-\rangle_x$. Note that the quantum simulator indicates that this amount of measurements is sufficient to estimate matrix elements of the density matrix with an error below 10^{-3} in the ideal noiseless case. In order to check the effect of decoherence, we apply a number of identity matrices (Figure 65, Right) which forces an additional waiting on the system, and hence promotes decoherence of the state. When applying 18 identity matrices, we obtain the following decomposition for $\widehat{\rho}$

$$\lambda_1 = 0.940 \rightarrow \begin{bmatrix} 0.727 - 0.032i \\ 0.686 \end{bmatrix} \qquad \lambda_2 = 0.060 \rightarrow \begin{bmatrix} 0.685 - 0.030i \\ -0.728 \end{bmatrix},$$

while application of 36 identity matrices results in

$$\lambda_1 = 0.927 \rightarrow \begin{bmatrix} 0.745 - 0.051i \\ 0.664 \end{bmatrix} \qquad \lambda_2 = 0.073 \rightarrow \begin{bmatrix} 0.663 - 0.045i \\ -0.747 \end{bmatrix}.$$

The effect of decoherence is visible in both more frequent occurence of the state that is close to $|-\rangle_x$, but also in the degradation of the eigenstates.

2. *Maximally entangled state for two qubits*

Let us now study the two-qubits maximally entangled state, which is an important part of all quantum algorithms achieving quantum speed-up over their classical counterparts. The state $\frac{1}{\sqrt{2}}(|10\rangle + |01\rangle)$ we are interested in is produced by the combination of H, X and $CNOT$ gates as shown in Figure 66, Left. We follow the same procedure as in the case of the Hadamard gate, described above, and first estimate the density matrix $\widehat{\rho}$ using 8152 measurements for each of the zz, yy, xx, zx and yz basis, and then decompose it as $\widehat{\rho} = \sum_{i=1}^{4} \lambda_i |\psi_i\rangle\langle\psi_i|$. Once again, ideally we should get $\lambda_1 = 1$ associated with $|\psi_1\rangle = \begin{bmatrix} 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \end{bmatrix}^T$. Instead, the analysis of the leading eigenvalues indicates that the eigenstate which is close (although significantly distorted) to the theoretical "ground truth" $|\psi_1\rangle$ above occurs in the mixture only with probability 0.87:

$$\lambda_1 = 0.871 \rightarrow \begin{bmatrix} -0.025 - 0.024i \\ 0.677 \\ 0.735 \\ -0.029 - 0.017i \end{bmatrix} \qquad \lambda_2 = 0.059 \rightarrow \begin{bmatrix} 0.598 \\ 0.123 + 0.468i \\ -0.075 - 0.445i \\ 0.454 - 0.022i \end{bmatrix}.$$

Our test of decoherence implemented using 18 identity matrices (see Figure 66, Right) shows that the probability of the "original" entangled state decreases to 0.79:

$$\lambda_1 = 0.793 \rightarrow \begin{bmatrix} -0.025 - 0.012i \\ 0.664 \\ 0.747 \\ -0.017 - 0.008i \end{bmatrix} \qquad \lambda_2 = 0.111 \rightarrow \begin{bmatrix} 0.997 \\ -0.002 - 0.058i \\ 0.035 + 0.036i \\ 0.006 + 0.007i \end{bmatrix}.$$

Interestingly enough, the second most probable eigenstate changes to the one that is close to $|00\rangle$. This might serve as an indication of the presence of biases in the machine.

D. Open problems and path forward

The application of the quantum tomography state reconstruction to simple states in the IBM QX revealed an important level of noise and decoherence present in the machine. It would be interesting to check if the states can be protected by using the error correction schemes. On the theoretical side, a number of problems related to quantum tomography still remain open. Those include the existence of optimal estimators, best choice of measurement operators (e.g., through active learning), and better reconstruction algorithms, in particular those that efficiently exploit the structure of the state and reduce the number of required measurements from exponential (in the general case) to polynomial. Exploration of these research questions are left for future work.

XXI. TESTS OF QUANTUM ERROR CORRECTION IN QUANTUM FINITE AUTOMATA

In this section, we use simplest quantum finite automata simulation to study whether quantum error correction (QEC) can improve computation accuracy. The answer to this question for IBM's 5-qubit chip is "No". Although some error correction effects are observed, improvements are not exponential and get completely spoiled by errors induced by extra gates and qubits needed for the error correction protocols.

A. Problem definition and background

It is commonly believed or stated that quantum algorithms are protected by quantum error correction (QEC) protocols. We refer the reader to a survey and introduction on QEC [33], while at the same time offering an alternative point of view that we support with a few experiments on the IBM chip. The idea of QEC is to entangle the qubit with several other ones and thus encode a quantum superposition state

$$|\psi\rangle = C_0|0\rangle + C_1|1\rangle, \tag{94}$$

using an entangled state, such as

$$|\psi\rangle = C_0|0\rangle^{\otimes n_q} + C_1|1\rangle^{\otimes n_q}, \tag{95}$$

where n_q is the number of qubits representing a single qubit in calculations.

The assumption is that small probability errors will likely lead to unwanted flips of only one qubit (in case when $n_q > 3$ this number can be bigger but we will not consider more complex situations here). Such errors produce states that are essentially different from those described by Equation (95). Measurements can then be used to fix a single qubit error using, e.g., through a majority voting strategy. More complex errors are assumed to be exponentially suppressed, which can be justified if qubits experience independent decoherence sources.

The idea of such QEC would probably work in quantum cryptography, where just transmission of a quantum state matters. However, we question whether QEC can work to protect quantum computations that require many quantum gate operations for the following reason: The main source of errors then is not spontaneous qubit decoherence but rather the finite fidelity of quantum gates. When quantum gates are applied to strongly entangled states, such as (95), they lead to *highly correlated* dynamics of *all* entangled qubits. We point out that errors introduced by such gates have essentially different nature from random uncorrelated qubit flips. So, gate-induced errors may not be treatable by standard error correction strategies.

To illustrate this point, imagine that we apply a gate that rotates a qubit by an angle $\pi/2$. It switches superposition states $|\psi_\pm\rangle = (|0\rangle \pm |1\rangle)/\sqrt{2}$ into, respectively, $|0\rangle$ or $|1\rangle$ in the measurement basis. Let the initial state be $|\psi_+\rangle$ but we do not know this before the final measurement. Initially, we know only that initial state can be either $|\psi_+\rangle$ or $|\psi_-\rangle$. To find what it is, we rotate qubit to the measurement basis. The gate is not perfect, so the final state after the gate application is

$$|u\rangle = \cos(\delta\phi)|0\rangle + \sin(\delta\phi)|1\rangle, \tag{96}$$

with some error angle $\delta\phi \ll 1$. Measurement of this state would produce wrong answer 1 with probability

$$P \approx (\delta\phi)^2. \tag{97}$$

The value $1 - P$ is called the fidelity of the gate. In IBM chip it is declared to be 0.99, which is not much. It means that after about 30 gates we should loose control. Error correction strategies can increase the number of allowed gates by an order of magnitude even at such a fidelity if we encode one qubit in three.

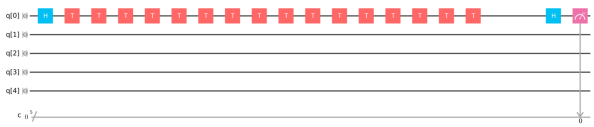


FIG. 67: Quantum circuit that creates state $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ then applies 16 T-gates that are equivalent to identity operation, and then applies the gate that transforms the entangled state into the trivial state $|0\rangle$.

In order to reduce this error, we can attempt to work with the 3-qubit version of the states (95). For example, let us consider the desired gate that transfers states

$$|\pm\rangle = |000\rangle \pm |111\rangle \tag{98}$$

into states $|000\rangle$ and $|111\rangle$ in the measurement basis, respectively. This gate is protected in the sense that a single unwanted random qubit flip leads to final states that are easily corrected by majority voting.

However, this is not enough because now we have to apply the gate that makes a rotation by $\pi/2$ in the basis (98). The error in this rotation angle leads to the final state

$$|u\rangle = \cos(\delta\phi)|000\rangle + \sin(\delta\phi)|111\rangle, \tag{99}$$

i.e., this particular error cannot be treated with majority voting because it flips all three qubits. On the other hand, this is precisely the type of errors that is most important when we have to apply many quantum gates because basic gate errors are mismatches between desired and received qubit rotation angles irrespectively of how qubits are encoded.

Based on these thoughts, traditional QEC cmay not be succeed in achieving exponential suppression of errors related to non-perfect quantum gate fidelity. The latter is the main source of decoherence in quantum computing that involves many quantum gates. As error correction is often called the only and first application that matters before quantum computing becomes viable at large scale, this problem e must be studied seriously and expeditiously. In the following subsection we report on our experimental studies of this problem with IBM's 5-qubit chip.

B. Test 1: errors in single qubit control

First, let us perform trivial operation shown in Fig. 67: we create a superposition of two qubit states

$$|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}, \tag{100}$$

then apply many gates that altogether do noting, i.e., they just bring the qubit back to the superposition state (100). We need those gates just to accumulate some error while qubit's state is not trivial in the measurement basis. Finally, we apply the gate that transforms its state back to $|0\rangle$.

Repeated experiments with measurements then produced wrong answer 13 times from 1000 samples. Thus, we estimate the error of the whole protocol, which did not use QEC, as

$$P_1 = 0.013,$$

or 1.6%. This is consistent and even better than declared 1% single gate fidelity because we applied totally 18 gates. So, either the used single qubit gates have much better fidelity than it is stated or this chip preprocesses submitted programs by concatenating repeated T-gates into a single pulse. Otherwise, the fidelity of a single gate looks to be closer to 99.9%.

C. Test 2: errors in entangled 3 qubits control

Next, we consider the circuit in Fig. 68 that initially creates the GHZ state $|-\rangle = (|000\rangle - |111\rangle)/\sqrt{2}$, then applies the same number, i.e. 16, of T-gates that lead to the same GHZ state. Then we apply the sub-circuit that brings the whole state back to $|000\rangle$.

Our goal is to quantify precision of identifying the final result with the state $|000\rangle$. If a single error bit flip happens, we can interpret results $|100\rangle$, $|010\rangle$ and $|001\rangle$ as $|000\rangle$ using majority voting. If needed, we can then apply a proper

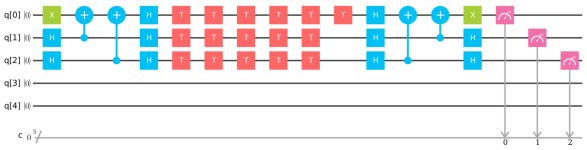


FIG. 68: Quantum circuit that creates state $|-\rangle = (|000\rangle - |111\rangle)/\sqrt{2}$ then applies 16 T-gates that are equivalent to identity operation, and then applies the gate that transforms the entangled GHZ state back into the trivial state $|000\rangle$. Measurements that return 1 for only one of the three qubits are interpreted as the $|000\rangle$ state at the end, while outcomes with two or three units are interpreted as the final state $|111\rangle$.

pulse to correct for this. So, in such cases we can consider the error treatable. If the total sum of probabilities of the final state $|000\rangle$ and final states with a single bit flipped is larger than P_1 from the previous single-qubit test, then we say that the quantum error correction works, otherwise, it doesn't. Our experiments showed that probabilities of events that lead to wrong final interpretation are as follows:

$$P_{110} = 0.006, \quad P_{101} = 0.02, \quad P_{011} = 0.016, \quad P_{111} = 0.005.$$

Thus, he probability to get the wrong interpretation of the result as the final state $|111\rangle$ of the encoded qubit is

$$P_3 = P_{110} + P_{101} + P_{011} + P_{111} = 0.047,$$

while the probability to get any error $1 - P_{000} = 0.16$.

D. Discussion

Comparing results of the tests without and with QEC, we find that implementation of the QEC does not improve the probability to interpret the final outcome correctly. The error probability of calculations without QEC gives a smaller probability of wrong interpretation, $P_1 = 1.3\%$, while the circuit with QEC gives an error probability $P_3 = 4.7\%$, even though we used majority voting that was supposed to suppress errors by about an order of magnitude.

More importantly, errors that lead to more than one qubit flip are not exponentially suppressed. For example, the probability $P_{101} = 0.02$ is close to the probability of a single bit flip event $P_{010} = 0.029$. We interpret this to mean that errors are not the results of purely random bit flip decoherence effects but rather follow from correlated errors induced by finite precision of quantum gates. The higher error rate in 3-qubit case could be attributed to the much worse fidelity of the controlled-NOT gate. However, even this error itself is small. The circuit itself produces the absolutely correct result $|000\rangle$ in 84% of simulations. If the remaining errors were produced by uncorrelated bit flips, we would see outcomes with more than one wrong bit flip with total probability less than 1% but we found that such events have a much larger total probability $P_3 = 4.7\%$.

In defense of QEC, we note, however, that probabilities of single bit flip errors where still several times larger than probabilities of multiple (two or three) wrong qubit flip errors. This means that at least partly QEC works, i.e., it corrects the state to $|000\rangle$ with 4.7% precision, versus the initially 16% in the wrong state. So, at least some part of the errors can be treated. However, efficient error correction must show *exponential* suppression of errors, which was not observed in this test.

Summarizing, this brief test shows no improvements that would be required for efficient quantum error correction. The need to use more quantum gates and qubits to correct errors only leads to larger probability of wrong interpretation of the final state. This problem will likely become increasingly much more important because without quantum error correction the whole idea of quantum computing is not practically useful. Fortunately, IBM's quantum chips can be used for experimental studies of this problem. We also would like to note that quantum computers can provide computational advantages beyond standard quantum algorithms and using only classical error correction [96]. So, they must be developed even if problems with quantum error correction prove detrimental for conventional quantum computing schemes.

XXII. CONCLUSION

In this article, we surveyed various quantum algorithms from the literature. We grouped these algorithms by their intended applications (see Table II). In each case, we gave the circuit diagram for the algorithm, and we showed the results from implementing this circuit on one of IBM's quantum computers. Our hope is that our algorithm implementations provide a blueprint that new quantum-computing users can follow to learn about quantum computing and to implement their own algorithms.

[1] Scott Aaronson and Lijie Chen. Complexity-theoretic foundations of quantum supremacy experiments. In Ryan O'Donnell, editor, *32nd Computational Complexity Conference, CCC 2017, July 6-9, 2017, Riga, Latvia*, volume 79 of *LIPIcs*, pages 22:1–22:67. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[2] A. Ambainis. Variable time amplitude amplification and a faster quantum algorithm for solving systems of linear equations. *ArXiv e-prints*, October 2010.

[3] A. Ambainis, H. Buhrman, P. Høyer, M. Karpinski, and P. Kurur. Quantum matrix verification. 2002.

[4] Andris Ambainis and R. Spalec. Quantum algorithms for matching and network flows. *in Lecture Notes in Computer Science: STACS 2006*, 3884, 2006.

[5] Itai Arad and Zeph Landau. Quantum computation and the evaluation of tensor networks. *SIAM Journal on Computing*, 39(7):3089–3121, 2010.

[6] Dave Bacon and Wim Van Dam. Recent progress in quantum algorithms. *Communications of the ACM*, 53(2):84–93, 2010.

[7] Stefanie Barz, Ivan Kassal, Martin Ringbauer, Yannick Ole Lipp, Borivoje Dakić, Alán Aspuru-Guzik, and Philip Walther. A two-qubit photonic quantum processor and its application to solving systems of linear equations. *Scientific reports*, 4, 2014.

[8] Robert Beals. Quantum computation of Fourier transforms over symmetric groups . *In Proceedings of STOC*, pages 48–53, 1997.

[9] Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, 26(5):1510–1523, 1997.

[10] E. Bernstein and U. Vazirani. Quantum complexity theory. In *Proc. of the Twenty-Fifth Annual ACM Symposium on Theory of Computing (STOC '93)*, pages 11–20, 1993. DOI:10.1145/167088.167097.

[11] Robin Blume-Kohout. Hedged maximum likelihood quantum state estimation. *Physical review letters*, 105(20):200504, 2010.

[12] Robin Blume-Kohout. Optimal, reliable estimation of quantum states. *New Journal of Physics*, 12(4):043034, 2010.

[13] Otakar Borůvka. O jistém problému minimálním. *Práce Mor. Přírodově d. spol. v Brnř (Acta Societ. Scient. Natur. Moravicae)*, 3:37–58, 1926.

[14] Michel Boyer et al. Tight bounds on quantum searching. *Fortsch.Phys.*, 58:493–506, 1998.

[15] G. Brassard et al. Quantum amplitude amplification and estimation. *Quantum Computation and Quantum Information*, 9, 2002.

[16] Bogislaw Broda. Quantum search of a real unstructured database. pages 1–7, 2015.

[17] H. Buhrman and R. Spalek. Quantum verification of matrix products. *arXiv.0409035*, 2004.

[18] X.-D. Cai, C. Weedbrook, Z.-E. Su, M.-C. Chen, M. Gu, M.-J. Zhu, L. Li, N.-L. Liu, C.-Y. Lu, and J.-W. Pan. Experimental Quantum Computing to Solve Systems of Linear Equations. *Physical Review Letters*, 110(23):230501, June 2013.

[19] Y. Cao, A. Daskin, S. Frankel, and S. Kais. Quantum circuit design for solving linear systems of equations. *Molecular Physics*, 110:1675–1680, August 2012.

[20] Jim Challenger, Andrew Cross, Vincent Dwyer, Mark Everittxi, Ismael Faro, Jay Gambetta, Juan Gomez, Paco Martin, Antonio Mezzacapo, Jesus Perez, Russell Rundle, Todd Tilma, John Smolin, Erick Winston, and Chris Wood. Quantum information software kit (QISKit). 2017.

[21] Kevin K. H. Cheung and Michele Mosca. Decomposing finite abelian groups. *Quantum Info. Comput.*, 1(3):26–32, October 2001.

[22] Andrew M Childs and Wim Van Dam. Quantum algorithms for algebraic problems. *Reviews of Modern Physics*, 82(1):1, 2010.

[23] Jill Cirasella. Classical and quantum algorithms for finding cycles. *MSc Thesis*, pages 1–58, 2006.

[24] C. Codsil and H. Zhan. Discrete-time quantum walks and graph structures. pages 1–37, 2011.

[25] Rigetti Computing. Quantum approximate optimization algorithm. Published online at `https://github.com/rigetticomputing/grove`, 2017. Accessed: 12/01/2017.

[26] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[27] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation*, (9):251–280, 1990.

[28] Marcus Cramer, Martin B Plenio, Steven T Flammia, Rolando Somma, David Gross, Stephen D Bartlett, Olivier Landon-Cardinal, David Poulin, and Yi-Kai Liu. Efficient quantum state tomography. *Nature communications*, 1:149,

2010.
[29] Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Vazirani. *Algorithms*. McGraw-Hill, Inc., New York, NY, USA, 2008.
[30] Aram W. Harrow Dave Bacon, Isaac L. Chuang. The Quantum Schur Transform: I. Efficient Qudit Circuits. pages 1–24, 2005.
[31] M. Dehn. Über unendliche diskontinuierliche gruppen. *Mathematische Annalen*, 71(1):116–144, Mar 1911.
[32] D. Deutsch and R. Jozsa. Rapid solutions of problems by quantum computation. In *Proc. of the Royal Society of London A*, pages 439–553, 1992. Bibcode:1992RSPSA.439..553D. DOI:10.1098/rspa.1992.0167.
[33] Simon J Devitt, William J Munro, and Kae Nemoto. Quantum error correction for beginners. *Reports on Progress in Physics*, 76(7):076001, 2013.
[34] B. L. Douglas and J. B. Wang. Efficient quantum circuit implementation of quantum walks. pages 1–6, 2009.
[35] Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006.
[36] Roman V. Dushkin. Quantum search using grover's algorithm. pages 1–10, 2015.
[37] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19 (2):248–264, 1972.
[38] Nayak F. Magniez A, J. Roland, and M. Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011.
[39] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014.
[40] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.
[41] R. Freivalds. Fast probabilistic algorithms. *In Proc. of 8th Symp. on Math. Foundations of Computer Science*, pages 57–69, 1979.
[42] Silvano Garnerone, Annalisa Marzuoli, and Mario Rasetti. Efficient quantum processing of 3-manifold topological invariants. *arXiv preprint quant-ph/0703037*, 2007.
[43] Joseph Geraci. A BQP-complete problem related to the Ising model partition function via a new connection between quantum circuits and graphs. *arXiv preprint arXiv:0801.4833*, 2008.
[44] Joseph Geraci and Daniel A Lidar. On the exact evaluation of certain instances of the Potts partition function by quantum computers. *Communications in Mathematical Physics*, 279(3):735–768, 2008.
[45] Craig Gidney. Grover's quantum search algorithm. 2013.
[46] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. 100:160501, 04 2008.
[47] David Gross, Yi-Kai Liu, Steven T Flammia, Stephen Becker, and Jens Eisert. Quantum state tomography via compressed sensing. *Physical review letters*, 105(15):150401, 2010.
[48] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.
[49] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009.
[50] Zdenek Hradil. Quantum-state estimation. *Physical Review A*, 55(3):R1561, 1997.
[51] IBM Corporation. IBM Quantum Experience. Published online at https://quantumexperience.ng.bluemix.net, 2016. Accessed: 12/01/2017.
[52] IBM Q Experience. Quantum composer. 2017.
[53] IBM Q Experience Documentation. Full user guide. 2017.
[54] IBM Q Experience Documentation. Grover's algorithm. 2017.
[55] Daniel F. V. James, Paul G. Kwiat, William J. Munro, and Andrew G. White. Measurement of qubits. *Phys. Rev. A*, 64:052312, 2001.
[56] Sonika Johri, Damian S. Steiger, and Matthias Troyer. Entanglement spectroscopy on a quantum computer. pages 1–7, 2017.
[57] Stephan Jordan. Quantum Algorithm Zoo. Published online at https://math.nist.gov/quantum/zoo/, 2011. Accessed: 3/18/2018.
[58] Stephen P. Jordan. Fast quantum algorithms for approximating some irreducible representations of groups . pages 1–21, 2009.
[59] J. Kempe. Quantum random walks - an introductory overview. *Contemporary Physics*, 44(4):307–327, 2003.
[60] V. Kendon. Where to quantum walk. pages 1–13, 2011.
[61] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. Adaptive Computation and Machine Learning. MIT Press, 2009.
[62] M W Krentel. The complexity of optimization problems. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, pages 69–76, New York, NY, USA, 1986. ACM.
[63] Thaddeus D Ladd, Fedor Jelezko, Raymond Laflamme, Yasunobu Nakamura, Christopher Monroe, and Jeremy Lloyd O?Brien. Quantum computers. *Nature*, 464(7285):45, 2010.
[64] François LeGall. An efficient quantum algorithm for some instances of the group isomorphism problem. *arXiv preprint arXiv:1001.0608*, 2010.
[65] R. J. Lipton and K. W. Regan. Quantum algorithms via linear algebra. 2014.
[66] S. Lloyd, M. Mohseni, and P. Rebentrost. Quantum algorithms for supervised and unsupervised machine learning. *ArXiv e-prints*, July 2013.
[67] Seth Lloyd, Silvano Garnerone, and Paolo Zanardi. Quantum algorithms for topological and geometric analysis of data.

Nature Communications, 2015.
[68] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631–633, 2014.
[69] N. B. Lovetti, S. Cooper, M. Everitt, M. Trevers, and V. Kendon. Universal quantum computation using discrete time quantum walk. pages 1–9, 2010.
[70] Frederic Magniez, Miklos Santha, and Mario Szegedy. Quantum algorithms for the triangle problem. *SIAM J. Comput.*, pages 413–424, 2007.
[71] E. Martín-López, A. Laing, T. Lawson, R. Alvarez, X.-Q. Zhou, and J. L. O'Brien. Experimental realization of Shor's quantum factoring algorithm using qubit recycling. *Nature Photonics*, 6:773–776, November 2012.
[72] Jarrod R. McClean, Jonathan Romero, Ryan Babbush, and AlÃąn Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2):023023, 2016.
[73] Ashley Montanaro. Quantum algorithms: an overview. *npj Quantum Information*, 2:15023, 2016.
[74] Michele Mosca. Quantum algorithms. In *Computational Complexity*, pages 2303–2333. Springer, 2012.
[75] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, New York, NY, USA, 10th edition, 2011.
[76] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, United Kingdom, 2016. 10th Anniversary Edition.
[77] M. A. Nielson and I. L. Chuang. *Quantum Computayion and Quantum Information (*10^{th} *Anniversary Edition)*. Cambridge University Press, New Delhi, India, 2010.
[78] J. Pan, Y. Cao, X. Yao, Z. Li, C. Ju, H. Chen, X. Peng, S. Kais, and J. Du. Experimental realization of quantum algorithm for solving linear systems of equations. *Phys. Rev. A*, 89(2):022313, February 2014.
[79] Karl Pearson. LIII. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 6*, 2(11):559–572, 1901.
[80] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alan Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5:ncomms5213, July 2014.
[81] Martin Plesch and Časlav Brukner. Quantum-state preparation with universal gate decompositions. *Phys. Rev. A*, 83:032302, Mar 2011.
[82] Carl Pomerance. A tale of two sieves. *Notices Amer. Math. Soc*, 43:1473–1485, 1996.
[83] Bo Qi, Zhibo Hou, Li Li, Daoyi Dong, Guoyong Xiang, and Guangcan Guo. Quantum state tomography via linear regression estimation. *Scientific reports*, 3, 2013.
[84] Rubens Viana Ramos, Paulo BenìĄcio Melo de Sousa, and David Sena Oliveira. Solving mathematical problems with quantum search algorithm. pages 1–9, 2006.
[85] Patrick Rebentrost, M Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. 113:130503, 09 2013.
[86] E. Rieffel and W. Polak. Quantum computing: A gentle introduction. 2011.
[87] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.
[88] Mehdi Saeedi and Igor L Markov. Synthesis and optimization of reversible circuitsâĂŤa survey. *ACM Computing Surveys (CSUR)*, 45(2):21, 2013.
[89] Miklos Santha. Quantum walk based search algorithms. In *International Conference on Theory and Applications of Models of Computation*, pages 31–46. Springer, 2008.
[90] N. Santhi. Quantum Netlist Compiler (QNC) software repository, November 2017. Applied for LANL LACC authorization for unlimited open-source release, December 2017.
[91] Jiangwei Shang, Zhengyun Zhang, and Hui Khoon Ng. Superfast maximum-likelihood reconstruction for quantum tomography. *Physical Review A*, 95(6):062336, 2017.
[92] Vivek V. Shende and Igor L. Markov. On the CNOT-cost of TOFFOLI gates. *Quant. Inf. Comp.*, 9(5-6):461–486, 2009.
[93] Peter W Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. IEEE, 1994.
[94] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.
[95] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.
[96] N. A. Sinitsyn. Computing with a single qubit faster than the quantum speed limit. *arXiv*, (arXiv:1701.05550), 2017.
[97] Robert S. Smith, Michael J. Curtis, and William J. Zeng. A practical quantum instruction set architecture, 2016.
[98] John A Smolin, Jay M Gambetta, and Graeme Smith. Efficient method for computing the maximum-likelihood quantum state from measurements with additive gaussian noise. *Physical review letters*, 108(7):070502, 2012.
[99] R. D. Somma. Quantum simulations of one dimensional quantum systems. *arXiv*, (arXiv:1503.06319v2 [quant-ph]), 2016.
[100] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, (13):354–356, 1969.
[101] J. A. K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Process. Lett.*, 9(3):293–300, June 1999.
[102] IBM QX Team. IBM Q experience backend information, November 2017. Last accessed: 12 December, 2017.
[103] Theoretical Computer Science Stack Exchange. Oracle construction for grover's algorithm. pages 1–3, 2017.
[104] Tommaso Toffoli. Reversible computing. *Automata, Languages and Programming*, pages 632–644, 1980.

[105] Giacomo Torlai, Guglielmo Mazzola, Juan Carrasquilla, Matthias Troyer, Roger Melko, and Giuseppe Carleo. Many-body quantum state tomography with neural networks. *arXiv preprint arXiv:1703.05334*, 2017.

[106] L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang. Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature (London)* , 414:883–887, December 2001.

[107] George F Viamontes, Igor L. Markov, and John P. Hayes. Graph-based simulation of quantum computation in the density matrix representation.

[108] Chu Ryang Wie. A quantum circuit to construct all maximal cliques using grover's search algorithm. pages 1–13, 2017.

[109] N. S. Yonofsky and M. A. Mannucci. Quantum computing for computer scientists. 2008.

[110] Yarui Zheng, Chao Song, Ming-Cheng Chen, Benxiang Xia, Wuxin Liu, Qiujiang Guo, Libo Zhang, Da Xu, Hui Deng, Keqiang Huang, et al. Solving systems of linear equations with a superconducting quantum processor. *Physical Review Letters*, 118(21):210504, 2017.