Quantum Algorithm Implementations for Beginners

ABHIJITH J., ADETOKUNBO ADEDOYIN, JOHN AMBROSIANO, PETR ANISIMOV, ANDREAS BÄRTSCHI, WILLIAM CASPER, GOPINATH CHENNUPATI, CARLETON COFFRIN, HRISTO DJIDJEV, DAVID GUNTER, SATISH KARRA, NATHAN LEMONS, SHIZENG LIN, ALEXANDER MALYZHENKOV, DAVID MASCARENAS, SUSAN MNISZEWSKI, BALU NADIGA, DANIEL O'MALLEY, DIANE OYEN, SCOTT PAKIN, LAKSHMAN PRASAD, RANDY ROBERTS, PHILLIP ROMERO, NANDAKISHORE SANTHI, NIKOLAI SINITSYN, PIETER J. SWART, JAMES G. WENDELBERGER, BORAM YOON, RICHARD ZAMORA, WEI ZHU, STEPHAN EIDENBENZ*, PATRICK J. COLES*, MARC VUFFRAY*, and ANDREY Y. LOKHOV*,

Los Alamos National Laboratory, Los Alamos, New Mexico 87545, USA

As quantum computers become available to the general public, the need has arisen to train a cohort of quantum programmers, many of whom have been developing classical computer programs for most of their careers. While currently available quantum computers have less than 100 qubits, quantum computing hardware is widely expected to grow in terms of qubit count, quality, and connectivity. This review aims to explain the principles of quantum programming, which are quite different from classical programming, with straightforward algebra that makes understanding of the underlying fascinating quantum mechanical principles optional. 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 succinct and self-contained fashion. We show how these algorithms can be implemented on IBM's quantum computer, and in each case, we discuss the results of the implementation with respect to differences between the simulator and the actual hardware runs. This article introduces computer scientists, physicists, and engineers to quantum algorithms and provides a blueprint for their implementations.

Contents

Abstract 1
Contents 1
1 Introduction 3
1.1 The quantum computing programming model 4
1.1.1 The qubit 4
1.1.2 System of qubits 5
1.1.3 Superposition and entanglement 5
1.1.4 Inner and outer products 6
1.1.5 Measurements 7
1.1.6 Unitary transformations and gates 7
1.1.7 Observables and expectation values 9
1.1.8 Quantum circuits 11
1.1.9 Quantum algorithms 12
1.2 Implementations on a real quantum computer 12
1.2.1 The IBM quantum computer 12
1.2.2 Programming the IBM quantum computer: Qiskit library 14
1.3 Classes of quantum algorithms 16
2 Grover's Algorithm 17
2.1 Problem definition and background 17

*eidenben@lanl.gov; pcoles@lanl.gov; vuffray@lanl.gov; lokhov@lanl.gov. LA-UR-20-22353

2.2 Algorithm description 18
2.3 Algorithm implemented on IBM's 5-qubit computer 19
3 Bernstein-Vazirani Algorithm 19
3.1 Problem definition and background 19
3.2 Algorithm description 20
3.3 Algorithm implemented on IBM's 5-qubit and 16-qubit computers 21
4 Linear Systems 23
4.1 Problem definition and background 23
4.2 Algorithm description 23
4.3 Phase estimation 24
4.4 Algorithm implemented on IBM's 5 qubit computer 26
5 Shor's Algorithm for Integer Factorization 28
5.1 Problem definition and background 28
5.2 Algorithm description 29
5.3 Algorithm implemented on IBM's 5-qubit computer 31
6 Matrix Elements of Group Representations 32
6.1 Problem definition and background 32
6.2 Algorithm description 36
6.3 Algorithm implemented on IBM's 5-qubit computer 37
7 Quantum Verification of Matrix Products 38
7.1 Problem definition and background 38
7.2 Algorithm description 38
8 Group Isomorphism 40
8.1 Problem definition and background 40
8.2 Algorithm description 40
8.3 Algorithm implemented using Qiskit 42
9 Quantum Persistent Homology 43
9.1 Problem definition and background 43
9.2 Quantum algorithm description 45
10 Quantum Random Walks 47
10.1 Problem definition and background 47
10.2 Example of a quantum random walk 48
10.3 Algorithm implementation using Qiskit on IBM Q 49
11 Quantum Minimal Spanning Tree 50
11.1 Problem definition and background 50
11.2 Algorithm description 51
12 Quantum Maximum Flow Analysis 54
12.1 Problem definition and background 54
12.2 Algorithm description 56
13 Quantum Approximate Optimization Algorithm 57
13.1 Problem definition and background 57
13.2 Algorithm description 58
13.3 QAOA MaxCut on `ibmqx2` 59
13.4 A proof-of-concept experiment 61
14 Quantum Principal Component Analysis 62
14.1 Problem definition and background 62
14.2 Algorithm description 63
14.3 Algorithm implemented on IBM's 5-qubit computer 65

15 Quantum Support Vector Machine 66
16 Quantum Simulation of the Schrödinger Equation 67
16.1 Problem definition and background 67
16.2 Algorithm description 68
16.3 Algorithm implemented on IBM's 5-qubit computer 69
17 Ground State of the Transverse Ising Model 70
17.1 Variational quantum eigenvalue solver 70
17.2 Simulation and results 72
18 Quantum Partition Function 74
18.1 Background on the partition function 74
18.2 A simple example 76
18.3 Calculating the quantum partition function 77
18.4 Implementation of a quantum algorithm on the IBM Quantum Experience 77
19 Quantum State Preparation 78
19.1 Single qubit state preparation 78
19.2 Schmidt decomposition 79
19.3 Two-qubit state preparation 81
19.4 Two-qubit gate preparation 81
19.5 Four qubit state preparation 82
20 Quantum Tomography 82
20.1 Problem definition and background 82
20.2 Short survey of existing methods 84
20.3 Implementation of the Maximum Likelihood method on 5-qubit IBM QX 85
20.3.1 Warm-up: Hadamard gate 85
20.3.2 Maximally entangled state for two qubits 86
21 Tests of Quantum Error Correction in IBM Q 87
21.1 Problem definition and background 87
21.2 Test 1: errors in single qubit control 88
21.3 Test 2: errors in entangled 3 qubits control 89
21.4 Discussion 89
Acknowledgments 90
References 90

1 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. Recently, Google announced that it has reached a major milestone known as quantum supremacy–the demonstration that a quantum computer can perform a calculation that is intractable on a classical supercomputer [8]. The problem tackled here by the quantum computer is not one with any direct real-world application. Nonetheless, this is a watershed moment for quantum computing and is widely seen as an important step on the road towards building quantum

computers that will offer practical speedups when solving real-world problems [82]. (See [2] 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 review, we provide a self-contained, succinct description of quantum computing and of the basic quantum algorithms with a focus on implementation. Since real quantum computers, such as IBM Q [55], 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 [10, 26, 58, 75, 76, 88]. Other cloud service based quantum computers are also available from Rigetti and IonQ, but in this review we will focus solely on IBM's quantum computing ecosystem. The code and implementations accompanying the paper can be found at https://github.com/lanl/quantum_algorithms.

1.1 The quantum computing programming model

Here we provide a self-contained description of the quantum computing programming model. We will define the common terms and concepts used in quantum algorithms literature. We will not discuss how the constructs explained here are related to the foundations of quantum mechanics. Interested readers are encouraged to take a look at Ref. [77] for a more detailed account along those lines. Readers with a computer science background are referred to Refs. [67, 85, 110], for a more comprehensive introduction to quantum computing from a computer science perspective.

Quantum computing basically deals with the manipulation of quantum systems. The physical details of this is dependent on the quantum computer's hardware design. Here we will only talk about the higher level abstractions used in quantum computing: a typical programmer will only be exposed to these abstractions. The state of any quantum system is always represented by a vector in a complex vector space (usually called a Hilbert space). Quantum algorithms are always expressible as transformations acting on this vector space. These basic facts follow from the axioms of quantum mechanics. Now we will explain some of the basic concepts and terminology used in quantum computing.

1.1.1 The qubit. The qubit (short for 'quantum bit') is the fundamental information carrying unit used in quantum computers. It can be seen as the quantum mechanical generalization of a bit used in classical computers. More precisely, a qubit is a two dimensional quantum system. The state of a qubit can be expressed as,

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

Here α and β are complex numbers such that, $|\alpha|^2 + |\beta|^2 = 1$. In the *ket-notation* or the *Dirac notation*, $|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}$ and $|1\rangle = \begin{pmatrix}0\\1\end{pmatrix}$ are shorthands for the vectors encoding the two basis states of a two dimensional vector space. So according to this notation, Eq. (1) expresses the fact that the state of the qubit is the two dimensional complex vector $\begin{pmatrix}\alpha\\\beta\end{pmatrix}$. Unlike a classical bit the state of a qubit cannot be measured without changing it. Measuring a qubit, whose state given by Eq. (1), will yield

the classical value of either zero ($|0\rangle$) with probability $|\alpha|^2$ or one ($|1\rangle$) with probability $|\beta|^2$. Qubit implementations and technologies are a very active area of research that is not the focus of our review, an interested reader is referred to [65] for a survey.

1.1.2 System of qubits. The mathematical structure of a qubit generalizes to higher dimensional quantum systems as well. The state of any quantum system is a normalized vector (a vector of norm one) in a complex vector space. The normalization is necessary to ensure that the total probability of all the outcomes of a measurement sum to one.

A quantum computer contains many number of qubits. So it is necessary to know how to construct the combined state of a system of qubits given the states of the individual qubits. The joint state of a system of qubits is described using an operation known as the *tensor product*, $\otimes$. Mathematically, taking the tensor product of two states is the same as taking the Kronecker product of their corresponding vectors. Say we have two single qubit states $|\phi\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix}$ and $|\phi'\rangle = \begin{pmatrix}\alpha'\\ \beta'\end{pmatrix}$. Then the full state of a system composed of two independent qubits is given by,

$$|\phi\rangle \otimes |\phi'\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix} \otimes \begin{pmatrix}\alpha'\\ \beta'\end{pmatrix} = \begin{pmatrix}\alpha\alpha'\\ \alpha\beta'\\ \beta\alpha'\\ \beta\beta'\end{pmatrix} \tag{2}$$

Sometimes the $\otimes$ symbol is dropped all together while denoting the tensor product to reduce clutter. Instead the states are written inside a single ket. For example, $|\phi\rangle \otimes |\phi'\rangle$ is shortened to $|\phi\phi'\rangle$, and $|0\rangle \otimes |0\rangle \otimes |0\rangle$ is shortened to $|000\rangle$. For larger systems the Dirac notation gives a more succinct way to compute the tensor product using the distributive property of the Kronecker product. For a system of, say, three qubits with each qubit in the state $\left|\gamma_j\right\rangle = \alpha_j |0\rangle + \beta_j |1\rangle$, for $j = 1, 2, 3$, the joint state is,

$$\begin{aligned} |\gamma_1\gamma_2\gamma_3\rangle &= |\gamma_1\rangle \otimes |\gamma_2\rangle \otimes |\gamma_3\rangle \qquad (3)\\ &= \alpha_1\alpha_2\alpha_3 |000\rangle + \alpha_1\alpha_2\beta_3 |001\rangle + \alpha_1\beta_2\alpha_3 |010\rangle + \alpha_1\beta_2\beta_3 |011\rangle \\ &\quad + \beta_1\alpha_2\alpha_3 |100\rangle + \beta_1\alpha_2\beta_3 |101\rangle + \beta_1\beta_2\alpha_3 |110\rangle + \beta_1\beta_2\beta_3 |111\rangle \qquad (4)\end{aligned}$$

A measurement of all three qubits could result in any of the eight (2^3) possible bit-strings associated with the eight basis vectors. One can see from these examples that the dimension of the state space grows exponentially in the number of qubits n and that the number of basis vectors is 2^n.

1.1.3 Superposition and entanglement. Superposition refers to the fact that any linear combination of two quantum states, once normalized, will also be a valid quantum state. The upshot to this is that any quantum state can be expressed as a linear combination of a few basis states. For example, we saw in Eq. (1) that any state of a qubit can be expressed as a linear combination of $|0\rangle$ and $|1\rangle$. Similarly, the state of any n qubit system can be written as a normalized linear combination of the 2^n bit-string states (states formed by the tensor products of $|0\rangle$'s and $|1\rangle$'s). The orthonormal basis formed by the 2^n bit-string states is called the *computational basis.*

Notice that Eq. (3) described a system of three qubits whose complete state was the tensor product of three different single qubit states. But it is possible for three qubits to be in a state that cannot be written as the tensor product of three single qubit states. An example of such a state is,

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle). \tag{5}$$

States of a system of which cannot be expressed as a tensor product of states of its individual subsystems are called *entangled states*. For a system of n qubits, this means that an entalged state

cannot be written a tensor product of n single qubit states. The existence of entangled states is a physical fact that has important consequences for quantum computing, and quantum information processing in general. In fact, without the existence of such states quantum computers would be no more powerful than their classical counterparts [108]. Entanglement makes it possible to create a complete 2^n dimensional complex vector space to do our computations in, using just n physical qubits.

1.1.4 Inner and outer products. We will now discuss some linear algebraic notions necessary for understanding quantum algorithms. First of these is the *inner product* or *overlap* between two quantum states. As we have seen before, quantum states are vectors in complex vectors spaces. The overlap between two states is just the inner product between these complex vectors. For example, take two single qubit states, $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$ and $|\psi\rangle = \gamma|0\rangle + \delta|1\rangle$. The overlap between these states is denoted in the ket notation as $\langle\psi|\phi\rangle$. And this is given by,

$$\langle\psi|\phi\rangle = \gamma^*\alpha + \delta^*\beta, \tag{6}$$

where * denotes the complex conjugate. Notice that,$\langle\psi|\phi\rangle = \langle\phi|\psi\rangle^*$. The overlap of two states is in general a complex number. The overlap of a state with a bit-string state will produce the corresponding coefficient. For instance from Eq. (1), $\langle 0|\phi\rangle = \alpha$ and $\langle 1|\phi\rangle = \beta$. And from Eq. (3), $\langle 001|\gamma_1\gamma_2\gamma_3\rangle = \alpha_1\alpha_2\beta_3$. Another way to look at overlaps between quantum states is by defining what is called a *bra* state. The states we have seen so far are ket states, like $|\phi\rangle$, which represented column vectors. A bra state corresponding to this ket state, written as $\langle\phi|$, represents a row vector with complex conjugated entries. For instance $|\phi\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix}$ implies that $\langle\phi| = \begin{pmatrix}\alpha^* & \beta^*\end{pmatrix}$. The overlap of two states is then the matrix product of a row vector with a column vector, yielding a single number. The reader must have already noticed the wordplay here. The overlap, with its closing angled parenthesis, form a 'bra-ket'!

The *outer product* of two states is an important operation that outputs a matrix given two states. The outer product of the two states we defined above will be denoted by, $|\psi\rangle\langle\phi|$. Mathematically the outer product of two states is a matrix obtained by multiplying the column vector of the first state with the complex conjugated row vector of the second state (notice how the ket is written before the bra to signify this). For example,

$$|\psi\rangle\langle\phi| = \begin{pmatrix}\alpha\\ \beta\end{pmatrix}\begin{pmatrix}\gamma^* & \delta^*\end{pmatrix} = \begin{pmatrix}\alpha\gamma^* & \alpha\delta^*\\ \beta\gamma^* & \beta\delta^*\end{pmatrix} \tag{7}$$

In this notation any matrix can be written as a linear combination of outer products between bit-string states. For a 2×2 matrix,

$$A = \begin{pmatrix}A_{00} & A_{01}\\ A_{10} & A_{11}\end{pmatrix} = A_{00}|0\rangle\langle 0| + A_{01}|0\rangle\langle 1| + A_{10}|1\rangle\langle 0| + A_{11}|1\rangle\langle 1|. \tag{8}$$

Acting on a state with a matrix then becomes just an exercise in computing overlaps between states. Let us demonstrate this process:

$$\begin{aligned} A|\phi\rangle &= A_{00}|0\rangle\langle 0|\phi\rangle + A_{01}|0\rangle\langle 1|\phi\rangle + A_{10}|1\rangle\langle 0|\phi\rangle + A_{11}|1\rangle\langle 1|\phi\rangle,\\ &= (A_{00}\alpha + A_{01}\beta)|0\rangle + (A_{10}\alpha + A_{11}\beta)|1\rangle = \begin{pmatrix}A_{00}\alpha + A_{01}\beta\\ A_{10}\alpha + A_{11}\beta\end{pmatrix}. \end{aligned} \tag{9}$$

This notation might look tedious at first glance but it makes algebraic manipulations of quantum states easily understandable. This is especially true when we are dealing with large number of qubits as otherwise we would have to explicitly write down exponentially large matrices.

The outer product notation for matrices also gives an intuitive input-output relation for them. For instance, the matrix $|0\rangle\langle 1| + |1\rangle\langle 0|$ can be read as "output 0 when given a 1 and output 1 when given a 0". Likewise,the matrix, $|00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 11| + |11\rangle\langle 10|$ can be interpreted as the mapping {"00" –> "00", "01" –> "01", "11" –> "10", "10" –> "11" }. But notice that this picture becomes a bit tedious when the input is in a superposition. In that case the correct output can be computed like in Eq. (9).

1.1.5 Measurements. Measurement corresponds to transforming the quantum information (stored in a quantum system) into classical information. For example, measuring a qubit typically corresponds to reading out a classical bit, i.e., whether the qubit is 0 or 1. A central principle of quantum mechanics is that measurement outcomes are *probabilistic*.

Using the aforementioned notation for inner products, for the single qubit state in Eq. (1), the probability of obtaining $|0\rangle$ after measurement is $|\langle 0|\phi\rangle|^2$ and the probability of obtaining $|1\rangle$ after measurement is $|\langle 1|\phi\rangle|^2$. So measurement probabilities can be represented as the squared absolute values of overlaps. Generalizing this, the probability of getting the bit string $|x_1 \ldots x_n\rangle$ after measuring an n qubit state, $|\phi\rangle$, is then $|\langle x_1 \ldots x_n|\phi\rangle|^2$.

Now consider a slightly more complex case of measurement. Suppose we have a three qubit state, $|\psi\rangle$ but we only measure the first qubit and leave the other two qubits undisturbed. What is the probability of observing a $|0\rangle$ in the first qubit? This probability will be given by,

$$\sum_{(x_2x_3)\in\{0,1\}^2} |\langle 0x_2x_3|\phi\rangle|^2. \tag{10}$$

The state of the system after this measurement will be obtained by normalizing the state,

$$\sum_{(x_2x_3)\in\{0,1\}^2} \langle 0x_2x_3|\phi\rangle\, |0x_2x_3\rangle\,. \tag{11}$$

Applying this paradigm to the state in Eq. (5) we see that the probability of getting $|0\rangle$ in the first qubit will be 0.5, and if this result is obtained, the final state of the system would change to $|000\rangle$. On the other hand, if we were to measure $|1\rangle$ in the first qubit we would end up with a state $|111\rangle$. Similarly we can compute the effect of subsystem measurements on any n qubit state.

In some cases we will need to do measurements on a basis different from the computational basis. This can be achieved by doing an appropriate transformation on the qubit register before measurement. Details of how to do this is given in a subsequent section discussing observables and expectation values.

The formalism discussed so far is sufficient to understand all measurement scenarios in this paper. We refer the reader to Ref. [77] for a more detailed and more general treatment of measurement.

1.1.6 Unitary transformations and gates. 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 called unitary if

$$UU^\dagger = U^\dagger U = I, \tag{12}$$

where $U^\dagger$ is the transposed, complex conjugate of U (called its *Hermitian conjugate*) 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

$$|\phi\rangle \rightarrow U|\phi\rangle = \begin{pmatrix} U_{00} & U_{01} \\ U_{10} & U_{11} \end{pmatrix}\begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \begin{pmatrix} U_{00}\alpha + U_{01}\beta \\ U_{10}\alpha + U_{11}\beta \end{pmatrix}. \tag{13}$$

Operators acting on different qubits can be combined using the Kronecker product. For example, if U_1 and U_2 are operators acting on two different qubits then the full operator acting on the combined two qubit system will be given by $U_1 \otimes U_2$.

For an n qubit system the set of physically allowed transformations, excluding measurements, consists of all $2^n \times 2^n$ unitary matrices. Notice that the size of a general transformation increases exponentially with the number of qubits. In practice a transformation on n qubits is effected by using a combination of unitary transformations that act only on one or two qubits at a time. By analogy to classical logic gates like NOT and AND, such basic unitary transformations, which are used to build up more complicated n qubit transformations, are called *gates*. Gates are unitary transformations themselves and from Eq. (12) it is clear that unitarity can only be satisfied if the number of input qubits is equal to the number of output qubits. Also, for every gate U it is always possible to have another gate $U^\dagger$ that undoes the transformation. So unlike classical gates quantum gates have to be reversible. *Reversible* means that the gate's inputs can always be reconstructed from the gate's outputs. For instance, a classical NOT gate, which maps 0 to 1 and 1 to 0 is reversible because an output of 1 implies the input was 0 and vice versa. However, a classical AND gate, which returns 1 if and only if both of its inputs are 1, is not reversible. An output of 1 implies that both inputs were 1, but an output of 0 provides insufficient information to determine if the inputs were 00, 01, or 10.

But this extra restriction of reversibility does not mean that quantum gates are 'less powerful' than classical gates. Even classical gates can be made reversible with minimal overhead. Reversibility does not restrict their expressive power [87]. Quantum gates can then be seen as a generalization of classical reversible gates.

The most common gates are described in Table 1. The X gate is the quantum version of the NOT gate. The CNOT or "controlled NOT" negates a target bit if and only if the control bit is 1. We will use the notation CNOT_{ij} for a CNOT gate controlled by qubit i acting on qubit j. The CNOT gate can be expressed using the outer product notation as,

$$\text{CNOT} = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X = |00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 11| + |11\rangle\langle 10|\,. \tag{14}$$

The Toffoli gate or "controlled-controlled NOT" or CCNOT, is a three qubit gate that is essentially the quantum (reversible) version of the AND gate. It negates a target bit if and only if both control bits are 1. In the outer product notation,

$$\text{CCNOT} = |11\rangle\langle 11| \otimes X + (I - |11\rangle\langle 11|) \otimes I. \tag{15}$$

Another way to look at the CCNOT gate is as a CNOT gate with an additional control qubit,

$$\text{CCNOT} = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes \text{CNOT}. \tag{16}$$

In general, one can define controlled versions of any unitary gate U as,

$$CU = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes U. \tag{17}$$

CU applies U to a set of qubits only if the first qubit (called the control qubit) is $|1\rangle$.

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 $\{H, T, \text{CNOT}\}$ (Refer Table 1 for definitions of these gates) forms a universal set. Also, the Toffoli gate by itself is universal [77].

One-qubit gates	Multi-qubit gates
Hadamard = $H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	CNOT = $CX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, $S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$	$CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$
$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$	Controlled-U = $CU = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & U_{00} & U_{01} \\ 0 & 0 & U_{10} & U_{11} \end{pmatrix}$
NOT = $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	SWAP = $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$, $Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	Toffoli (CCNOT) = $\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$
$R(\theta) = P(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$	

Table 1. Commonly used quantum gates.

1.1.7 Observables and expectation values. We have seen that experiments in quantum mechanics are probabilistic. Often in experiments we will need to associate a real number with a measurement outcome. And quantities that we measure in quantum mechanics will always be statistical averages of these numbers. For instance, suppose we do the following experiment on many copies of the single qubit state in Eq. (1): We measure a copy of the state and if we get $|0\rangle$ we record 1 in our lab notebook , otherwise we record -1. While doing this experiment we can never predict the outcome of a specific measurement. But we can ask statistical questions like: "What will be the average value of the numbers in the notebook?" From our earlier discussion on measurement we know that the probability of measuring $|0\rangle$ is $|\alpha|^2$ and the probability of measuring $|1\rangle$ is $|\beta|^2$. So the average value of the numbers in the notebook will be,

$$|\alpha|^2 - |\beta|^2 \tag{18}$$

In quantum formalism, there is neat way to express such experiments and their average outcomes, without all the verbiage, using certain operators. For the experiment described above the associated operator would be the Z gate,

$$Z = |0\rangle\langle 0| - |1\rangle\langle 1| = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{19}$$

By associating this operator with the experiment we can write the average outcome of the experiment, on $|\phi\rangle$, as the overlap between $|\phi\rangle$ and $Z|\phi\rangle$,

$$\langle\phi|Z|\phi\rangle = \langle\phi|0\rangle\langle 0|\phi\rangle - \langle\phi|1\rangle\langle 1|\phi\rangle = |\alpha|^2 - |\beta|^2. \tag{20}$$

The operator Z is called the *observable* associated with this experiment. And the quantity $\langle\phi|Z|\phi\rangle$ is called its *expectation value*. The expectation value is sometimes denoted by $\langle Z\rangle$, when there is no ambiguity about the state on which the experiment is performed.

Here we saw an experiment done in the computational basis. But this need not always be the case. Experiments can be designed by associating real numbers to measurement outcomes in any basis. What would be the observable for such an experiment? For an experiment that associates the real numbers $\{a_i\}$ to a measurement onto a basis set $\{|\Phi_i\rangle\}$, the observable will be,

$$O \equiv \sum_i a_i |\Phi_i\rangle\langle\Phi_i| . \tag{21}$$

This observable will reproduce the correct expectation value for this experiment done on any state $|\psi\rangle$,

$$\langle\psi|O|\psi\rangle = \sum_i a_i \langle\psi|\Phi_i\rangle\langle\Phi_i|\psi\rangle = \sum_i a_i |\langle\Phi_i|\psi\rangle|^2. \tag{22}$$

Because the states $\{|\Phi_i\rangle\}$ are orthonormal, we can see that O obeys the following eigenvalue equation,

$$O|\Phi_j\rangle = \sum_i a_i |\Phi_i\rangle\langle\Phi_i|\Phi_j\rangle = a_j|\Phi_j\rangle . \tag{23}$$

So O is an operator that has complete set of orthogonal eigenvectors and real eigenvalues. Such operators are called *Hermitian operators*. Equivalently, these operators are equal to their Hermitian conjugates ($O = O^\dagger$). In quantum mechanics, any Hermitian operator is a valid observable. The eigenvectors of the operator give the possible outcomes of the experiment and the corresponding eigenvalues are the real numbers associated with that outcome.

But can all valid observables be measured in practice? The answer to this depends on the quantum system under consideration. In this tutorial, the system under consideration is an IBM quantum processor. And in these processors only measurements onto the computational basis are supported natively. Measurements to other basis states can be performed by applying an appropriate unitary transformation before measurement. Suppose that the hardware only lets us do measurements onto the computational basis $\{|i\rangle\}$ but we want to perform a measurement onto the basis set $\{|\Phi_i\rangle\}$. This problem can be solved if we can implement the following unitary transformation,

$$U = \sum_i |i\rangle\langle\Phi_i| . \tag{24}$$

Now measuring $U|\psi\rangle$ in the computational basis is the same as measuring $|\psi\rangle$ in the $\{|\Phi_i\rangle\}$ basis. This can be seen by computing the outcome probabilities on $U|\psi\rangle$,

$$|\langle j|U|\psi\rangle|^2 = |\sum_i \langle j|i\rangle\langle\Phi_i|\psi\rangle|^2 = |\langle\Phi_j|\psi\rangle|^2. \tag{25}$$

So once U is applied, the outcome $|j\rangle$ becomes equivalent to the outcome $\left|\Phi_j\right\rangle$ in the original measurement scenario. Now, not all such unitary transformations are easy to implement. So if a quantum algorithm requires us to perform a measurement onto some complicated set of basis states, then the cost of implementing the corresponding U has be taken into account.

1.1.8 Quantum circuits. Quantum algorithms are often diagrammatically represented as circuits in literature. Here we will describe how to construct and read quantum circuits. In the circuit representation, qubits are represented by horizontal lines. Gates are then drawn on the qubits they act on. This is done in sequence from left to right. The initial state of the qubit is denoted at the beginning of each of the qubit lines. Notice that when we write down a mathematical expression for the circuit, the gates are written down from right to left in the order of their application.

These principles are best illustrated by an example. Given in Fig. 1 is a circuit to preparing an entangled two qubit state called a Bell state from $|00\rangle$.

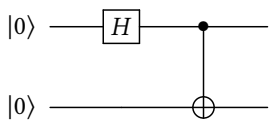


Fig. 1. Quantum circuit for preparing a Bell state

The circuit encodes the equation,

$$\mathrm{CNOT}_{12}\,(H \otimes I)\,|00\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

Let us now carefully go over how the circuit produces the Bell state. We read the circuit from left to right. The qubits are numerically labelled starting from the top. First the H gate acts on the top most qubit changing the state of the system to,

$$H \otimes I\,|00\rangle = (H\,|0\rangle) \otimes (I\,|0\rangle) = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) \otimes |0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle).$$

Then CNOT_{12} acts on both of these qubits. The blackened dot on the first qubit implies that this qubit is the control qubit for the CNOT. The $\oplus$ symbol on the second qubit implies that this qubit is the target of the NOT gate (controlled by the state of the first qubit). The action of the CNOT then gives,

$$\mathrm{CNOT}_{12}\left(\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)\right) = \frac{1}{\sqrt{2}}(\mathrm{CNOT}_{12}\,|00\rangle + \mathrm{CNOT}_{12}\,|10\rangle) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

The measurement of a qubit is also denoted by a special gate with a meter symbol on it, given in Fig 2. The presence of this gate on a qubit means that the qubit must be measured in the computational basis.

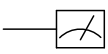


Fig. 2. The measurement gate

1.1.9 Quantum algorithms. We have now introduced all the basic elements needed for the discussion of practical quantum algorithms. A quantum algorithm consists of three basic steps:

- Encoding of the data, which could be classical or quantum, into the state of a set of input qubits.
- A sequence of quantum gates applied to this set of input qubits.
- Measurements of one or more of the qubits at the end to obtain a classically interpretable result.

In this review, we will describe the implementation of these three steps for a variety of quantum algorithms.

1.2 Implementations on a real quantum computer

1.2.1 The IBM 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 other quantum processors freely accessible through the IBM Quantum Experience platform. These processors can be accessed by visiting the IBM Quantum Experience website (https://quantum-computing.ibm.com/)

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 the `ibmqx4`, the available gates include:

$$\{I, X, Y, Z, H, S, S^\dagger, T, T^\dagger, U_1(\lambda), U_2(\lambda,\phi), U_3(\lambda,\phi,\theta), \text{CNOT}\}. \tag{26}$$

The Graphical User Interface (GUI) also provides other controlled gates and operations like measurement and reset. Most of these gates appear in our Table 1. The gates $U_1(\lambda)$, $U_2(\lambda,\phi)$, and $U_3(\lambda,\phi,\theta)$ are continuously parameterized gates, defined as follows:

$$U_1(\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix}, \quad U_2(\lambda,\phi) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i(\lambda+\phi)} \end{pmatrix}, \quad U_3(\lambda,\phi,\theta) = \begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i(\lambda+\phi)}\cos(\theta/2) \end{pmatrix}. \tag{27}$$

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

The gates listed in Eq. (26) 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 (26) into products of gates from a physical gate set. The physical gate set employed by IBM is essentially composed of three gates [1]:

$$\{U_1(\lambda), R_X(\pi/2), \text{CNOT}\}. \tag{28}$$

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:

$$R_X(\pi/2) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}. \tag{29}$$

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

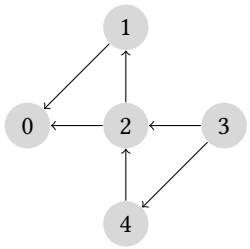


Fig. 3. The connectivity diagram of `ibmqx4`. The circles represent qubits and the arrows represent the ability to apply a physical CNOT gate between the qubits.

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 (Fig.3). 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.,

$$\mathrm{CNOT}_{kj} = (H \otimes H)\mathrm{CNOT}_{jk}(H \otimes H)\,. \tag{30}$$

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:

$$\mathrm{CNOT}_{jl} = \mathrm{CNOT}_{kl}\mathrm{CNOT}_{jk}\mathrm{CNOT}_{kl}\mathrm{CNOT}_{jk}\,. \tag{31}$$

Hence, using (30) and (31), 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 gates do not precisely correspond to the physically implemented gates. 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. Decoherence refers to the fact that gradually over time the quantum computer loses its "quantumness" and behaves 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. The error rates for individual qubits in the IBM processors are listed in the IBM Quantum Experience website. In this tutorial, we will show in many cases how infidelity and decoherence affect the algorithm performance in practice.

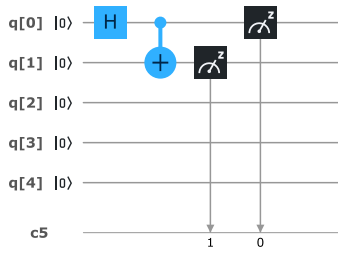


Fig. 4. The quantum circuit to prepare a Bell state and measure it in the IBM quantum experience GUI

A simple example of programming the IBM quantum computer is given in Fig. 4, which shows the Bell state preparation circuit Fig.1 compiled using the IBM quantum experience GUI. Extra measurement operations at the end serve to verify the fidelity of the implementation.

1.2.2 Programming the IBM quantum computer: Qiskit library. Qiskit [3] is an open-source quantum computing library developed under the aegis of IBM. Qiskit allows users to write and run programs on either IBM's quantum processors or on a local simulator, without the use of the graphical interface. This is an important feature because the graphical interface becomes impractical as the number qubits become large. At the time of writing, users can use Qiskit to access quantum processors with up to 16 qubits. Smaller processors are also accessible. Qiskit is a very powerful software development kit (SDK) which has multiple elements in it that tackle a variety of problems associated with practical quantum computing. Qiskit is further split into four modules called: Terra, Aer, Aqua, and Ignis. Each of these modules deal with a specific part of quantum software development. In this section we will only give a brief overview of programming simple quantum circuits with Qiskit. For a comprehensive overview of Qiskit and its various capabilities, the reader is encouraged to visit the official website (www.qiskit.org) [3].

For our purposes, Qiskit can be viewed as a Python library for quantum circuit execution. A basic Qiskit code has two parts, designing the circuit and running it. In the circuit design phase, we create an instance of `QuantumCircuit` with the required number of qubits and classical bits. Then gates and measurements are added to this blank circuit. Gates and measurements are implemented in Qiskit as methods of the `QuantumCircuit` class. After the circuit has been designed we need to choose a backend to run the circuit. This can be either be a simulator called the `qasm_simulator` or it can be one of IBM's quantum processors. To use a quantum processor, you will need to load your IBM Q account information into Qiskit. Given in Fig. 5 is a simple code to construct the Bell state. This is the Qiskit version of the circuit in Fig. 1 with measurement added at the end to verify our results.

```
### Quantum circuit for preparing the Bell state ####

import numpy as np
from qiskit import QuantumCircuit, execute, Aer

# Create a Quantum Circuit with two qbits and 2 classical bits
circuit = QuantumCircuit(2,2)

# Add a H gate on qubit 0
circuit.h(0)

# Add a CX (CNOT) gate on control qubit 0 and target qubit 1
circuit.cx(0,1)

# Map the quantum measurement to the classical bits
circuit.measure([0,1],[0,1])


# Use Aer's qasm_simulator
simulator = Aer.get_backend('qasm_simulator')

# Execute the circuit on the qasm simulator
job = execute(circuit, simulator, shots=1000)

# Grab results from the job
result = job.result()

# Returns counts
counts = result.get_counts(circuit)
print("\nTotal count for 00 and 11 are:",counts)
```

Fig. 5. Qiskit code to create and measure a Bell state. Source: www.qiskit.org

In Fig.5 we are running the circuit on the simulator for 1000 independent runs. The final output was `{'11': 493, '00': 507}`. This is what we expect from measuring the Bell state $(\frac{|00\rangle+|11\rangle}{\sqrt{2}})$, up to statistical fluctuations. While running the same code on the 14 qubit `ibmq_16_melbourne` processor for 1024 runs gave $|11\rangle$ with probability 0.358 and $|00\rangle$ with probability 0.54. The remaining probability was distributed over 01 and 10, which should not be a part of the Bell state. As we discussed before, this phenomenon is due to errors inherent to the quantum processor. As the backend technology improves we expect to get better results from these trials. Often, we will also present a circuit using OpenQASM (Open Quantum Assembly Language). OpenQASM provides an intermediate representation of a program in the form of a quantum circuit, that is neither the actual program written by the programmer nor the machine instructions seen by the processor. OpenQASM 'scores' we show in this paper will be simple sequence of gates and measurements, with the corresponding registers that they act on. The syntax of these scores will be self explanatory.

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	ESSEX	low
	Matrix Product Verification	GO	n.a.	n.a.
	Subgroup Isomorphism	QFT	none	n.a.
	Persistent Homology	GO, QFT	QX4	med-low
Graph Applications	Quantum Random Walk	n.a.	VIGO	med-low
	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	Schrödinger 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 2. 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

1.3 Classes of quantum algorithms

In this review, we broadly classify quantum algorithms according to their area of application. We will discuss quantum algorithms for graph theory, number theory, machine learning and so on. The complete list of algorithms discussed in this paper, classified according to their application areas, can be found in Table 2. The reader is also encouraged to take a look at the excellent Quantum Algorithm Zoo website [58] for a concise and comprehensive list of quantum algorithms.

In classical computing, algorithms are often designed by making use of one or more algorithmic paradigms like dynamic programming or local search, to name a few. Most known quantum algorithms also use a combination of algorithmic paradigms specific to quantum computing. 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 number of known quantum algorithmic paradigms is much smaller compared to the number of known classical paradigms. The constraint of unitarity on quantum operations and the impossibility of non-intrusive measurement make it difficult to design quantum paradigms from existing classical paradigms. But researchers are constantly in search for new paradigms and we can expect this list to get longer in the future. Table 2 also contains information about the paradigms used by the algorithms in this article.

The rest of the paper presents each of the algorithms shown in Table 2, 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[1].

[1]The code and implementations for most of the algorithms can be found at https://github.com/lanl/quantum_algorithms.

2 GROVER'S ALGORITHM

2.1 Problem definition and background

Grover's algorithm as initially described [52] 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 [14] 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 one could construct a "quantum oracle" 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 for two integers $1 < a < b$ such that $ab = n$ for some number n, resulting in a factoring algorithm. Grover's search in this case would have worse performance than Shor's algorithm [93, 94] described below, which is a specialised algorithm to solve the factoring problem. 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 $\mathbf{x}$. We use the term *ancilla* or *ancillary qubit* to refer to some extra qubits that are used by the algorithm.

The function $f(\mathbf{x})$ is defined by

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} = \mathbf{x}^* \\ 0 & \text{if } \mathbf{x} \neq \mathbf{x}^* \end{cases} \tag{32}$$

where $\mathbf{x} = x_1 x_2 \ldots x_n$ are binary strings and $\mathbf{x}^*$ is the specific string 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. In essence, Grover's algorithm can invert an arbitrary function with binary outputs, provided we have a quantum oracle that implements the function. Grover's algorithm has been used, with appropriate oracles, to solve problems like finding triangles in a graph [72], finding cycles [28], and finding maximal cliques [109]. For the analysis of Grover's algorithm, the internals of the oracle is typically considered a black-box. Often, the oracle operator for the problem at hand has to be constructed as a quantum circuit. But, keep in mind that an inefficient oracle construction can nullify any practical advantages gained by using Grover's search.

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. But AND gate is not reversible and cannot be a quantum gate. However the Toffoli gate, that was introduced in the previous section, is a reversible version of the classical AND gate. 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. The unitary matrix corresponding to the Toffoli gate can be found in Table 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 $|\mathbf{x}\rangle |q\rangle \rightarrow |\mathbf{x}\rangle \left|f(\mathbf{x}) \bigoplus q\right\rangle$, where $\bigoplus$ is the XOR operation . Here we will only discuss the case where $\mathbf{x}^*$ is unique. Grover's algorithm can also be used to search for multiple items in a database.

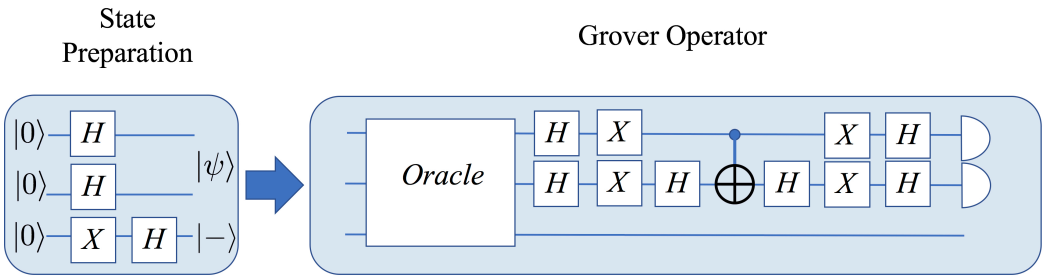


Fig. 6. A schematic diagram of Grover's algorithm is shown. Note that in this case, one application of the Grover operator is performed. This is all that is necessary when there are only two bits in $\mathbf{x}$, but the Grover operator should be applied repeatedly for larger problems.

2.2 Algorithm description

Here we present a brief introduction to Grover's algorithm. A more detailed account can be found in Nielsen and Chuang [77]. Let N be the number of items (represented as bit strings) amongst which we are performing the search. This number will also be equal to the dimension of the vector space we are working with. An operator, called the Grover operator or the diffusion operator, is the key piece of machinery in Grover's algorithm. This operator is defined by

$$G = (2\left|\psi\right\rangle\left\langle\psi\right| - I)O \tag{33}$$

where $\left|\psi\right\rangle = \frac{1}{\sqrt{N}}\sum_i \left|i\right\rangle$ is the uniform superposition over all the basis states and O is the oracle operator (see Fig. 6 for a representation of this operator in the case where $\mathbf{x}$ consists of 2 bits). The action of $(2\left|\psi\right\rangle\left\langle\psi\right| - I)$ on an arbitrary state, given by $\sum_i a_i \left|i\right\rangle$, when decomposed over the basis states is,

$$(2\left|\psi\right\rangle\left\langle\psi\right| - I)\sum_i a_i \left|i\right\rangle = \sum_i (2\left\langle a\right\rangle - a_i)\left|i\right\rangle \tag{34}$$

where $\left\langle a\right\rangle = \frac{\sum_i a_i}{N}$ is the average amplitude in the basis states. From Eq. (34) one can see that the amplitude of each $\left|i\right\rangle$-state (a_i) is flipped about the mean amplitude.

In order to use the Grover operator to successfully perform a search, the qubit register must be appropriately initialized. The initialization is carried out by applying a Hadamard transform to each of the the main qubits ($H^{\otimes n}$) and applying a Pauli X transform followed by a Hadamard transform (HX) to the ancilla. This leaves the main register in the uniform superposition of all states, $\left|\psi\right\rangle$, and the ancilla in the state $\frac{\left|0\right\rangle-\left|1\right\rangle}{\sqrt{2}}$. After performing these operations, the system is in the state $\left|\psi\right\rangle\frac{\left|0\right\rangle-\left|1\right\rangle}{\sqrt{2}}$. Using Eq. (34), we can now understand how the Grover operator works. The action of the oracle operator on $\left|\mathbf{x}^*\right\rangle\frac{\left|0\right\rangle-\left|1\right\rangle}{\sqrt{2}}$ reverses the amplitude of that state

$$O\left|\mathbf{x}^*\right\rangle\frac{\left|0\right\rangle-\left|1\right\rangle}{\sqrt{2}} \rightarrow \left|\mathbf{x}^*\right\rangle\frac{\left|f(\mathbf{x}^*)\bigoplus 0\right\rangle - \left|f(\mathbf{x}^*)\bigoplus 1\right\rangle)}{\sqrt{2}} = \left|\mathbf{x}^*\right\rangle\frac{\left|1\right\rangle-\left|0\right\rangle}{\sqrt{2}} = -\left|\mathbf{x}^*\right\rangle\frac{\left|0\right\rangle-\left|1\right\rangle}{\sqrt{2}} \tag{35}$$

A similar argument shows that all other states are unmodified by the oracle operator. Combining this with Eq. (34) 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 $\left|\mathbf{x}^*\right\rangle$ is below $\left\langle a\right\rangle$ (using the notation of Eq. (34)) while all the other states have an amplitude that is slightly above $\left\langle a\right\rangle$. The effect of applying $2\left|\psi\right\rangle\left\langle\psi\right| - I$ is then to make $\left|\mathbf{x}^*\right\rangle$ 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 exactly $\left\lceil \frac{\pi\sqrt{N}}{4} \right\rceil$ times after which a measurement will reveal $\mathbf{x}^*$ with probability close to 1. 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). Below is a high level pseudocode for the algorithm.

Algorithm 1 Grover's algorithm

Input:

- An Oracle operator effecting the transformation $|x\rangle\,|q\rangle \rightarrow |x\rangle\,|q \oplus f(x)\rangle$.

Output:

- The unique bit string $\mathbf{x}^*$ satisfying Eq. (32)

Procedure:

Step 1. Perform state initialization $|0\ldots0\rangle \rightarrow |\psi\rangle\,(\frac{|0\rangle-|1\rangle}{\sqrt{2}})$

Step 2. Apply Grover operator $\left\lceil \frac{\pi\sqrt{N}}{4} \right\rceil$ times

Step 3. Perform measurement on all qubit except the ancillary qubit.

2.3 Algorithm implemented on IBM's 5-qubit computer

Fig. 7 shows the circuit that was designed to fit the `ibmqx4` quantum computer. The Toffoli gate is not available directly in `ibmqx4` so it has to be constructed from the available set of gates given in Eq. 26.

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. 7) as the ancillary qubit, and $q[1]$ and $q[2]$ as x_1 and x_2 respectively. Note that the quantum computer imposes constraints on the possible source and target of CNOT gates.

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 due to the inherent errors in `ibmqx4`. This deviation will get worse for circuits of larger size.

We also ran another test using CNOT gates that did not respect the underlying connectivity of the computer. This resulted in a significantly deeper circuit and the results were inferior to the results with the circuit in Fig. 7.

This implementation used a Toffoli gate with a depth of 23 (compared to a depth of 13 here) and obtained the correct answer 48% of the time.

3 BERNSTEIN-VAZIRANI ALGORITHM

3.1 Problem definition and background

Suppose we are given a classical Boolean function, $f : \{0, 1\}^n \mapsto \{0, 1\}$. It is guaranteed that this function can always be represented in the form, $f_{\mathbf{s}}(\mathbf{x}) = \bigoplus_i s_i x_i \equiv \langle \mathbf{s}, \mathbf{x} \rangle$. Here, $\mathbf{s}$ is an unknown bit string, which we shall call a *hidden string*. Just like in Grover's algorithm we assume that we have a quantum oracle that can compute this function.

The Bernstein-Vazirani (BV) algorithm then finds the hidden string with just a single application of the oracle. The number of times the oracle is applied during an algorithm algorithm is known as

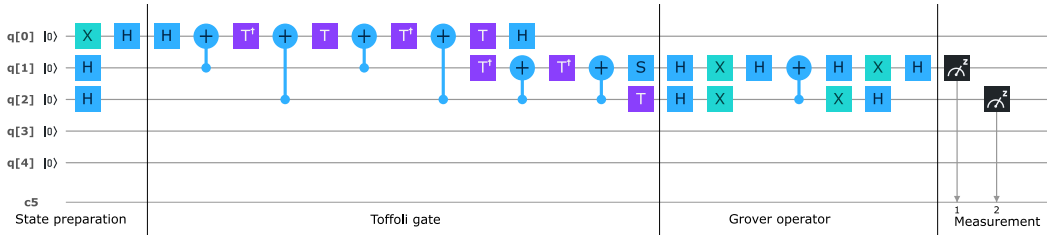


Fig. 7. The circuit that was executed on IBM's 5-qubit quantum computer. The first two time slots correspond to the state preparation. The next 13 time slots implement a Toffoli gate. The next 7 time slots implement the $2|\psi\rangle\langle\psi| - I$ operator, and the final two time slots are used for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

its *query complexity*. The BV algorithm has a query complexity of one. From our earlier discussions we saw that Grover's algorithm has a query complexity of $O(\sqrt{N})$.

In the classical case each call to $f_{\mathbf{s}}(\mathbf{x})$ 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.

The quantum algorithm to solve this problem was developed by Bernstein and Vazirani [15] building upon the earlier work of Deutsch and Jozsa [36]. Their contribution was a quantum algorithm for the hidden string problem, which has a non-recursive quantum query complexity of just 1. This constitutes a polynomial $O(n)$ query-complexity separation between classical and quantum computation. They also discovered a less widely known recursive hidden-string query algorithm, which shows a $O(n^{\log n})$ 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 computational power of quantum computers.

One thing to note about the BV 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 $O(4^n)$[77]. Since the black box is a step in the algorithm, its serial execution time could in the worst-case even scale exponentially. The real breakthrough of this quantum algorithm lies in speeding up the query complexity and not the execution time per se.

3.2 Algorithm description

Let us explore the BV algorithm in more detail. Let U_s be the oracle for the function $f_{\mathbf{s}}(\mathbf{x})$. It acts in the usual way and computes the value of the function onto an ancilla qubit,

$$U_s|\mathbf{x}\rangle|q\rangle = |\mathbf{x}\rangle|q \oplus \langle \mathbf{s}, \mathbf{x}\rangle\rangle \tag{36}$$

Denoting $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$, we can easily verify from Eq. (35) that,

$$U_s|\mathbf{x}\rangle|-\rangle = (-1)^{\langle \mathbf{s},\mathbf{x}\rangle}|\mathbf{x}\rangle|-\rangle\,. \tag{37}$$

Also, note that the n-qubit Hadamard operator, which is just n single qubit H operators applied in parallel, can be expanded as,

$$H^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x},\mathbf{y}\in\{0,1\}^n} (-1)^{\langle \mathbf{x},\mathbf{y}\rangle}|\mathbf{y}\rangle\langle\mathbf{x}| \tag{38}$$

The reader may verify this identity by applying $H^{\otimes n}$ to the computational basis states.

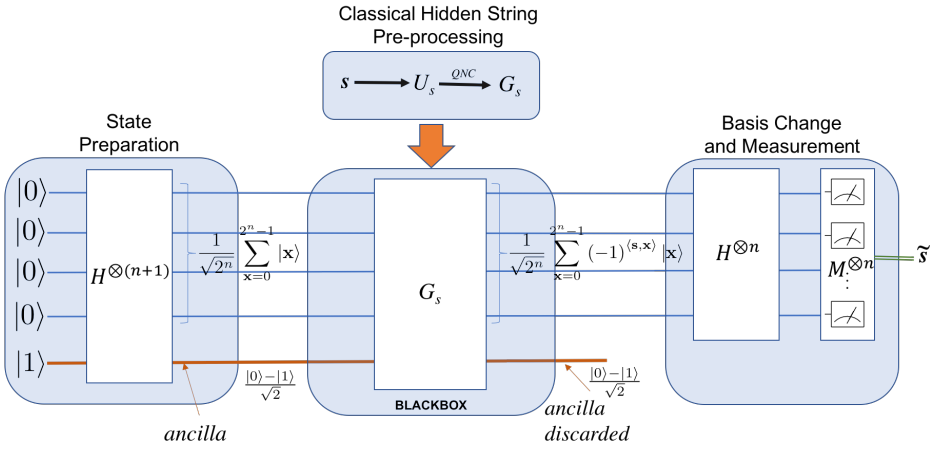


Fig. 8. Bernstein-Vazirani hidden string discovery quantum algorithm. The hidden string **s** is discovered with just a single query. The measurement result $\widetilde{\mathbf{s}}$ gives the hidden string.

U_s and $H^{\otimes n}$ are the only two operators needed for the BV algorithm. The pseudocode for the algorithm is given in Algorithm 2. Notice that the initialization part is identical to that of Grover's algorithm. This kind of initialization is a very common strategy in quantum algorithms.

Algorithm 2 Bernstein-Vazirani algorithm

Input:
- An oracle operator, U_s, effecting the transformation $|x\rangle\,|q\rangle \rightarrow |x\rangle\,|q \oplus \langle \mathbf{s}, \mathbf{x}\rangle\rangle$.

Output:
- The hidden string **s**.

Procedure:

Step 1. Perform state initialization on $n+1$ qubits, $|0\ldots 0\rangle \rightarrow |\psi\rangle\,|-\rangle$

Step 2. Apply U_s .

Step 3. Apply $H^{\otimes n}$ to the first n qubits.

Step 4. Measure all qubits except the ancillary qubit.

The final measurement will reveal the hidden string, **s**, with probability 1. Let us now delve into the algorithm to see how this result is achieved. The entire circuit for the BV algorithm is represented in Figure 8. This circuit can be analyzed as follows,

$$
\begin{aligned}
|0\rangle^n\,|1\rangle \xrightarrow{H^{\otimes(n)}\otimes H} & \frac{1}{\sqrt{2^n}}\sum_{\mathbf{x}=0}^{2^n-1}|\mathbf{x}\rangle\otimes|-\rangle \xrightarrow{U_s} \frac{1}{\sqrt{2^n}}\sum_{\mathbf{x}=0}^{2^n-1}(-1)^{\langle \mathbf{s},\mathbf{x}\rangle}\,|\mathbf{x}\rangle\otimes|-\rangle \\
\xrightarrow{H^{\otimes n}} & \frac{1}{\sqrt{2^n}}\sum_{\mathbf{x},\mathbf{y}=0}^{2^n-1}(-1)^{\langle \mathbf{s},\mathbf{x}\rangle\oplus\langle \mathbf{x},\mathbf{y}\rangle}\,|\mathbf{y}\rangle\otimes|-\rangle \quad \equiv \quad |\mathbf{s}\rangle\otimes|-\rangle\,. \qquad (39)
\end{aligned}
$$

Here we have crucially used the identity for $H^{\otimes n}$ given in Eq.(38).

3.3 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 oracle U_s given a binary hidden

string **s**. Let us see how this is done using an example binary hidden string "01". Equation (40) below shows how the 3-qubit operator maps the $2^3 = 8$ basis vectors onto themselves. The first line is the input binary vector (in the order x_1, x_0, q), and the second line is the output binary vector.

$$U_{01} = \begin{pmatrix} 000 & 010 & 100 & 110 & 001 & 011 & 101 & 111 \\ 000 & 011 & 100 & 111 & 001 & 010 & 101 & 110 \end{pmatrix} \tag{40}$$

This mapping, $U_{01} : |\mathbf{x}\rangle\,|q\rangle \mapsto |\mathbf{x}\rangle\,|\langle 01, \mathbf{x}\rangle \oplus q\rangle$, 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 given in Eq (26). The time cost of applying these gates can be accessed from IBM's published calibration models [103] for the primitive hardware gates.

In order to decompose arbitrary unitary matrices to the primitive gates, we need to first perform a unitary diagonalization of the $2^{(n+1)} \times 2^{(n+1)}$ matrix using multi-qubit-controlled single-qubit unitary Given's rotation operations. Such multi-qubit-controlled single-qubit operations can be decomposed further to primitive gates using standard techniques [77] to the hardware primitive gates. Even after this step we will be left with arbitrary CNOT gates that do not respect the topology of the underlying quantum processor. Since both `ibmqx4`, `ibmqx5` computers have restricted CNOT connectivity between qubits, we will need to decompose the CNOT gates further into available CNOT gates using the method discussed in the introductory section. As we saw in the Grover's algorithm section, such decompositions will further degrade the quality of our results. As the overall primitive gate counts scale as $O(4^n)$ for arbitrary n-qubit unitary operators, these decompositions quickly becomes hard to do by hand. To address this we wrote a piece of software called *Quantum Netlist Compiler (QNC)* [89] for performing these transformations automatically. 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 8 for a specific bit-string **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[2] 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× 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 (circuit) generated by QNC for U_{01} for `ibmqx4` is shown in Figure 9. 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.

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 10. 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.

[2] These times are estimated using the data available from IBM at the time of writing. These values will change as the hardware improves.

Even for 2-bit strings, the scores were quite long, and the results were quite noisy even with 8192 machine-shots.

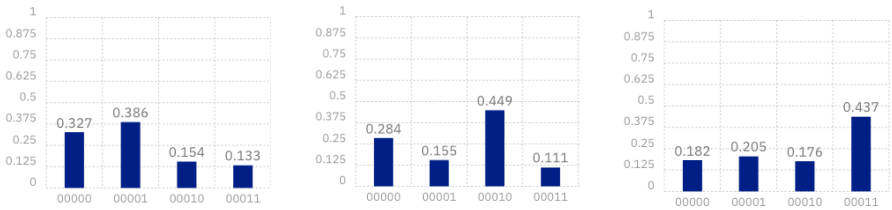


Fig. 10. Results from running the BV algorithm for 8192 shots on 2-bit hidden-strings "01", "10" and "11" respectively (left to right) on `ibmqx4`. The y-axis here is the probability of obtaining the hidden string, which theoretically should be 1.

4 LINEAR SYSTEMS

4.1 Problem definition and background

Solving linear systems is central to a majority of science, engineering, finance and economics applications. For example, one comes across such systems while solving differential or partial differential equations or while performing regression. The problem of solving a system of linear equations is the following: 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

$$A\,|x\rangle = |b\rangle\,, \tag{41}$$

with the solution $|x\rangle$ being

$$|x\rangle = \frac{A^{-1}\,|b\rangle}{\|A^{-1}\,|b\rangle\,\|}. \tag{42}$$

4.2 Algorithm description

The quantum algorithm for the linear system was first proposed by Harrow, Hassidim, and Lloyd (HHL) [53]. The HHL algorithm has been implemented on various quantum computers in [11, 24, 111]. 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 computer, 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.

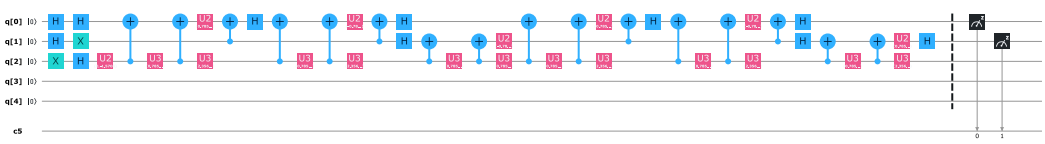


Fig. 9. Quantum circuit for BV algorithm with hidden string "01" targeting the `ibmqx4` architecture.

The main idea of the algorithm is as follows. Let $\{|u_j\rangle\}$ and $\{\lambda_j\}$ be the eigenvectors 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 eigenvectors $\{|u_j\rangle\}$, $|b\rangle = \sum_{j=1}^{N} \beta_j |u_j\rangle$. 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:

$$R^\dagger \Lambda R |x\rangle = |b\rangle \overset{\text{Step1}}{\implies} \Lambda R |x\rangle = R |b\rangle \overset{\text{Step2}}{\implies} R |x\rangle = \Lambda^{-1} R |b\rangle \overset{\text{Step3}}{\implies} |x\rangle = R^\dagger \Lambda^{-1} R |b\rangle \tag{43}$$

This procedure requires us to find the eigenvalues of A. This can be done using a quantum subroutine called *phase estimation.* We will discuss this subroutine in some detail as it is a common ingredient in many quantum algorithms.

4.3 Phase estimation

Phase estimation is a quantum subroutine that lets us find the eigenvalues of a unitary matrix U given the ability to apply it to a quantum register as a controlled gate. Let $|u\rangle$ be an eigenvector of U such that, $U |u\rangle = e^{2\pi i \lambda_u} |u\rangle$. Then the phase estimation subroutine effects the following transformation,

$$|0\rangle |u\rangle \rightarrow \left|\tilde{\lambda}_u\right\rangle |u\rangle . \tag{44}$$

Here $\tilde{\lambda}_u$ is an estimate for λ_u. This subroutine makes use of an important transformation called the Quantum Fourier Transform (QFT)

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

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

where the ij-th element of the matrix is $W_{ij} = \omega^{ij}$ and ω is a primitive N-th root of one($\omega^N = 1$). 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 11. The Quantum Fourier Transform (QFT) is defined as a transformation between two 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 the QFT is defined as the transformation

$$QFT \left(\sum_{k=0}^{N-1} x_k |k\rangle \right) = \sum_{k=0}^{N-1} y_k |k\rangle . \tag{45}$$

The implementation of the QFT mimics the stages (recursive calls) of the 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 12.

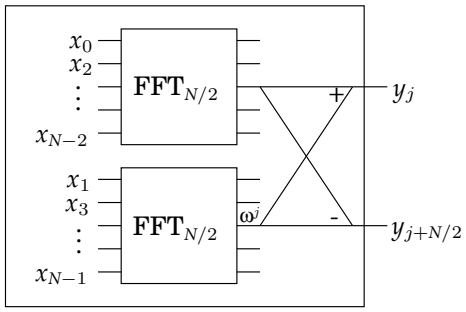


Fig. 11. Fast Fourier Transform circuit, where j denotes a row from the top half of the circuit and ω^j denotes that the corresponding value is multiplied by ω^j. The plus and minus symbols indicate that the corresponding values have to be added or subtracted, respectively.

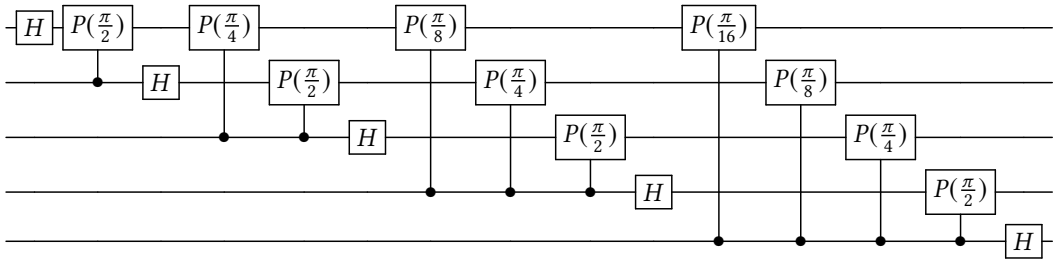


Fig. 12. A Quantum Fourier Transform circuit for five qubits ($n = 5$).

The phase estimation procedure cleverly uses the QFT operator to estimate the eigenphases of the operator U. The circuit for performing phase estimation given in Fig. 13. Notice that the QFT is applied in reverse.

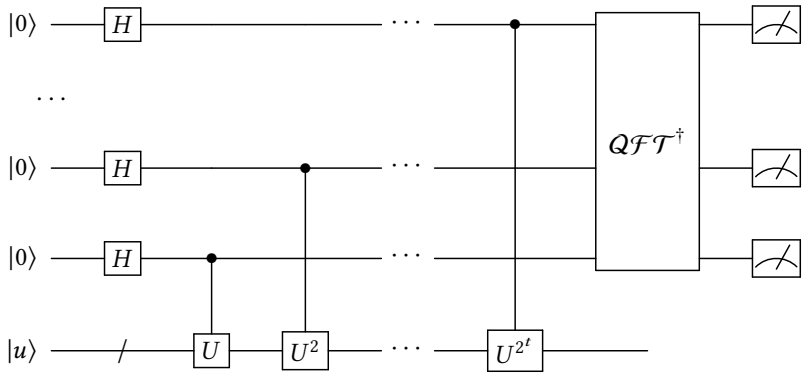


Fig. 13. Quantum circuit for phase estimation.

The pseudocode for phase estimation is given in Algorithm 3. Notice that the algorithm also works if the input state is not an eigenstate. The output in this case can be determined by expanding the input state in terms of the eigenstates and then applying the linearity of quantum operations.

In the code, we have numbered the ancillary qubits from the top and C_iU denotes the unitary controlled by the i^{th} ancilla qubit acting on the main n qubit register.

Algorithm 3 Phase estimation subroutine

Input:

- Controlled unitaries C_iU
- An n qubit input state $|\psi\rangle = \sum_u \psi_u |u\rangle$, where $U|u\rangle = e^{2\pi i\lambda_u}|u\rangle$.

Output:

- $\sum_u \psi_u \left|\tilde{\lambda}_u\right\rangle |u\rangle$

Procedure:

Step 1. Take t ancillary qubits initialized to zero and perform $H^{\otimes t}$ on them to produce the uniform superposition state over them.

for $0 \leq i < t$ **do**

Step 2. Apply $C_{t-i-1}U^{2^i}$

end for

Step 3. Apply $\mathcal{QFT}^\dagger$.

Optional Measure the ancillary qubits to get $\left|\tilde{\lambda}_u\right\rangle |u\rangle$ with probability $|\psi_u|^2$

The number of ancillary qubits used in the phase estimation algorithm will determine both its run-time and its accuracy. On the accuracy front, the number of ancillary qubits used is equal to the bit precision of $\tilde{\lambda}_u$ as the answer is stored in this register. The exact complexity of this subroutine is discussed in Ref. [77].

Now we can discuss the HHL algorithm which makes use of the phase estimation procedure to perform a matrix inversion. The HHL algorithm 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 [111], and we found it to be most useful in keeping things clear. The HHL algorithm requires us to run the phase estimation procedure on the unitary operator e^{iA}. The phases estimated would be approximations to the eigenvalues of A. The problem of applying the unitary operation e^{iA} given the matrix A is called *quantum simulation*. There are many algorithms in literature that tackle the problem of quantum simulation [16] [47] and that will not be our focus in this section. We will explain the steps of the HHL algorithm below assuming that the quantum simulation part is taken care of. We will also include some mathematical details in the pseudocode given in Algorithm 4 .

These three steps are equivalent to the three steps shown in Eq. (43). The algorithm is probabilistic, we get $|x\rangle$ only if the final measurement gives $|1\rangle$. But this probability can be boosted using a technique called *amplitude amplification* [21]. This technique is explained in detail in Section VII.

4.4 Algorithm implemented on IBM's 5 qubit computer

Now we implement 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 use 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 $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix} \equiv |-\rangle$ and

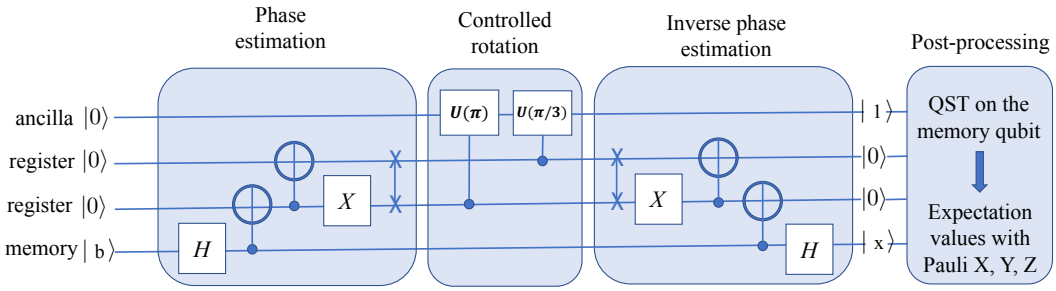


Fig. 14. Schematic of the circuit for the quantum algorithm for solving a 2×2 linear system. The first step involves phase estimation, which maps the eigenvalues λ_j of A into the register in the binary form. The second step involves controlled rotation of the ancilla qubit, so that the inverse of the eigenvalues $\frac{1}{\lambda_j}$ show up 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.

Algorithm 4 HHL algorithm

Input:

- The state $|b\rangle = \sum_j \beta_j |u_j\rangle$
- The ability to perform controlled operations with unitaries of the form e^{iAt}

Output:

- The quantum state $|x\rangle$ such that $A\vec{x} = \vec{b}$.

Procedure:

Step 1. Perform quantum phase estimation using the unitary transformation e^{iA}. This maps the eigenvalues λ_j into the register in the binary form to transform the system,

$$|0\rangle_a |0\rangle_r |b\rangle_m \rightarrow \sum_{j=1}^{N} \beta_j |0\rangle_a |\lambda_j\rangle_r |u_j\rangle_m . \tag{46}$$

Step 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

$$\sum_{j=1}^{N} \beta_j \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle_a + \frac{C}{\lambda_j} |1\rangle_a \right) |\lambda_j\rangle_r |u_j\rangle_m . \tag{47}$$

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

$$\sum_{j=1}^{N} \beta_j \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle_a + \frac{C}{\lambda_j} |1\rangle_a \right) |0\rangle_r |u_j\rangle_m . \tag{48}$$

Step 4. Measuring the ancilla qubit will give ,

$$|x\rangle \approx \sum_{j=1}^{N} C \left(\frac{\beta_j}{\lambda_j} \right) |u_j\rangle , \tag{49}$$

if the measurement outcome is $|1\rangle$

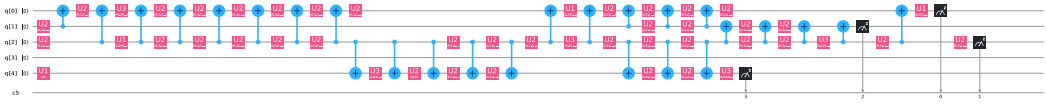


Fig. 15. Circuit implemented on IBM's 5-qubit ibmqx4 quantum computer for the case with $|b\rangle$ set to $|0\rangle$ and with $\langle Z \rangle$ measurement. After implementing the circuit in Fig. 14 and setting the coupling map of the ibmqx4 architecture, Qiskit-sdk-py re-arranges the qubits to fit the mapping. This circuit represents the outcome of the re-arrangement which was implemented on the ibmqx4 quantum computer.

Table 3. 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 $\langle Z \rangle$
$\|0\rangle$	-0.60	-0.60	0.00	-0.027	0.80	0.81
$\|+\rangle$	1.00	1.00	0.00	-0.06	0.00	0.02
$\|-\rangle$	-1.00	-1.00	0.0060	0.000	-0.02	0.00

$\frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} \equiv |+\rangle$, respectively. For this system, the three steps of the HHL algorithm, can be performed by the operations shown in Fig. 14. For the controlled rotation, we use 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. (47). Both λ and ϕ are set to zero in these controlled U rotations. Although the composer on Quantum Experience does not have this gate, in IBM Qiskit-sdk-py, we use cu3 function for this purpose. Three cases are 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 $\langle X \rangle$, $\langle Y \rangle$, $\langle Z \rangle$ can then be performed to obtain the expectation values. QASM code is output from Qiskit-sdk-py and then uploaded on to IBM Quantum Experience. Figure 15 shows the equivalent composer circuit generated from QASM for the measurement in the computational basis (Z measurement).

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 3. The simulator expectation values and the theoretical values match well. This shows that the implementation of the algorithm gives expected results. Similar expectation values were also seen using the simulator on IBM Quantum Experience instead of the local simulator. We then ran the circuit on the quantum computer ibmqx4. Fig. 16 shows a comparison between the simulator results and the results from the ibmqx4 with Z measurement on the circuit. As can be seen from Fig. 16, the results from the actual run do not give the expected answer as seen in the simulator results. We remark that recent modifications to the algorithm [22, 101] can in some cases allow for larger scale and more accurate implementations on noisy quantum computers.

5 SHOR'S ALGORITHM FOR INTEGER FACTORIZATION

5.1 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

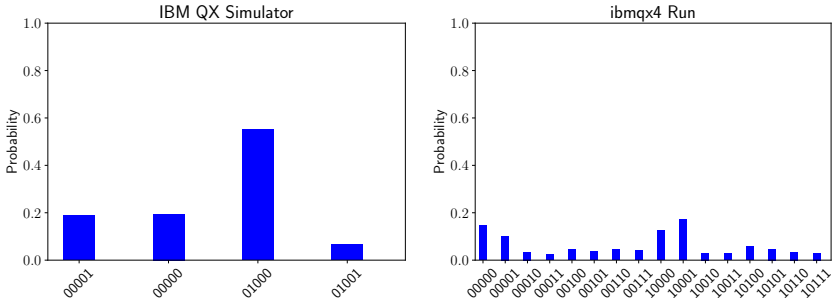


Fig. 16. Results of the circuit with Z measurement (computational basis measurement) from the actual run and the simulator on a `ibmqx4`. 4096 shots were used for both the cases.

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 classical algorithm has running time $O\left(\exp\left(\sqrt[3]{\frac{64}{9}n(\log n)^2}\right)\right)$ [81]. 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 [86], 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 experimental implementation of this 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 [73].

In this section we describe Shor's algorithm and its implementation on `ibmqx4`

5.2 Algorithm description

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

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

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 a p, one can find the factors of N efficiently as gcd can be computed in polynomial time.

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

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

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). The QFT operation was introduced earlier during our discussion of phase estimation.

The property of the 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 $QFT\left(\sum_{i=0}^{M} x_i \, |i\rangle\right) = \sum_{i=0}^{M} y_i \, |i\rangle$, 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 in Fig 17 [34]. The first QFT on register A produces an equal superposition of the qubits from A, i.e., the resulting

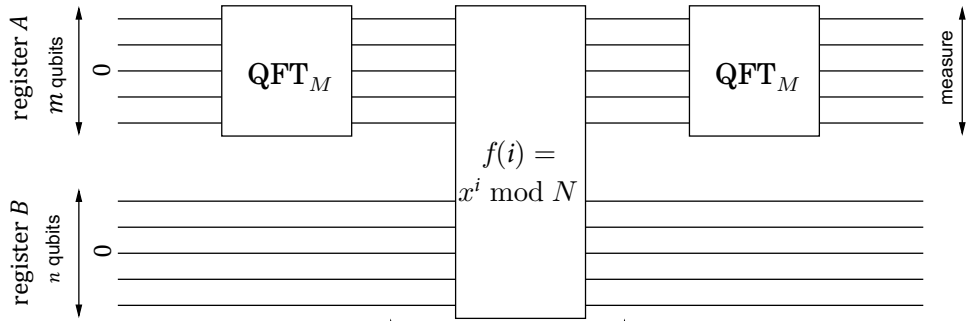


Fig. 17. Illustration of the period-finding circuit, where $m = 2n$ and $M = 2^m$.

state is

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

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

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

Before we apply the next QFT transform, we do a measurement of register B. (By the principle of deferred measurement [77] 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 [77].

Period finding can also be viewed as a special case of phase estimation. The reader may refer Nielsen and Chuang [77] for this perspective on period finding.

5.3 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 17 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 (Fig 18).

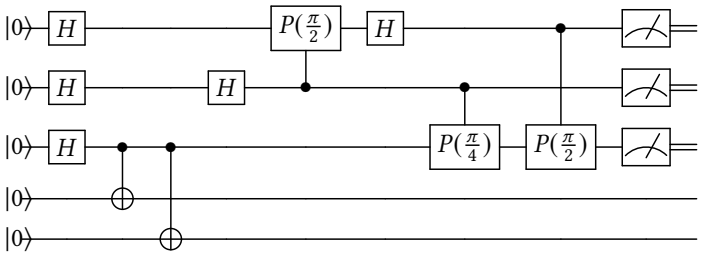


Fig. 18. Circuit for Shor's algorithm for $N = 15$ and $x = 11$.

The results from the measurements are shown on Figure 19.

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.

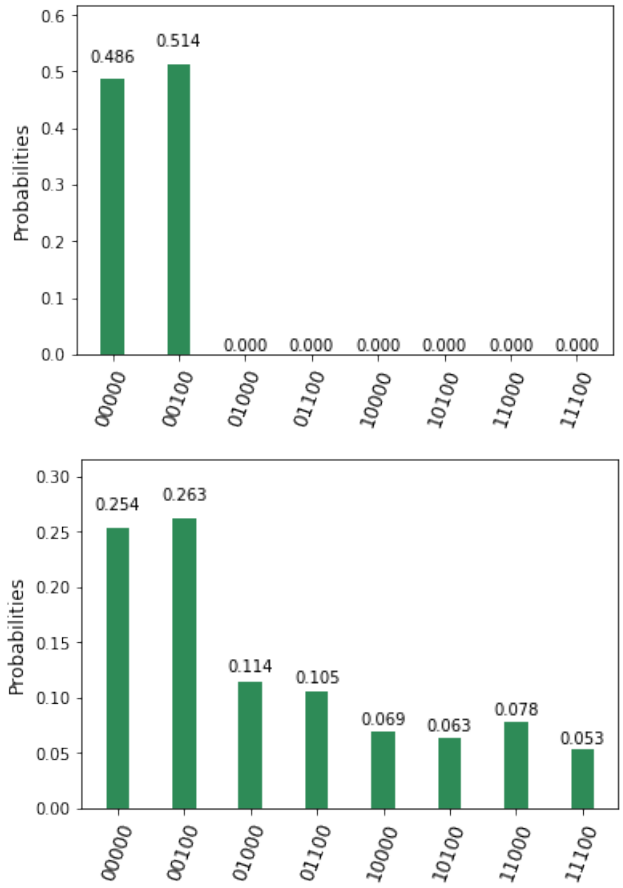


Fig. 19. Output from the circuit from Figure 18 implemented on the simulator (left) and `ibmqx4` (right).

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

6 MATRIX ELEMENTS OF GROUP REPRESENTATIONS

6.1 Problem definition and background

In this section we will discuss another quantum algorithm that makes use of the QFT operation. In this section we will also introduce a subroutine called the *Hadamard test*, which lets us compute matrix elements of unitary operators. But first, we will require some knowledge of group theory to understand the problem being tackled here. This section follows the work of Jordan in Ref. [59].

A *Group* $(G, \cdot)$ or (G) is a mathematical object defined by its elements $(g_1, g_2, \ldots)$ and an operation between elements $(\cdot)$, such that these 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 $\forall\ g_i \in G$, there 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 section, we will discuss quantum algorithms to solve certain problems related to finite groups. As before, we will also implement them on the IBM machines. Some examples of groups are given below.

Example 1A. Abelian group A_n with n elements: $0, 1, \ldots, n-1$, and the group operation addition modulo n: $g_i \cdot g_j = (i+j)\text{mod}(n)$. For instance, for $n = 3$: $a_0 = 0$, $a_1 = 1$, $a_2 = 2$. Then, $a_2 \cdot a_2 = 4$ mod(3) $= 1 = a_1$, $a_2 \cdot a_1 = 3$ mod(3) $= 0 = a_0$, *etc.* The identity element is $a_0 = 0$ and its inverse is itself. For all other elements 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: $[1, 2.., n]$, $[2, 1.., n]$, $\ldots$, $[n, n-1.., 2, 1]$. 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: $ab \rightarrow ab$, while one permutation p is the second group element: $ab \rightarrow ba$. 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] $\rightarrow$ [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 operations between them is not trivial and/or the group order, n, is large. In this case, it is helpful to apply the representation theory to the group. The idea is simple: if we can map a group of unknown objects with nontrivial operations to the group of known objects with some trivial operations, we can gain some information about the unknown group. In general, we introduce a function applied to a group element: $\rho\,(g_i)$, which does this mapping between two groups. 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: $a_j \rightarrow \rho(a_j) = e^{i2\pi j/N}$, where the original operation (+mod(n)) is substituted by the new operation of multiplication. Note that the group S_2 can be represented in the same way as A_2.

Example 2S. Representation of group S_3: $s_j \rightarrow \rho(s_j) = 1$, 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 is, $[1, 1, 1, -1, -1, -1] \rightarrow [s_1, s_2, \ldots, s_n]$, where we map odd permutations to -1 and even permutations to 1 . While it carries more information about the initial group than the trivial representation, it does not imply that the group S_3 is not Abelian. One cannot construct 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.

Most useful representations are often ones which map a group to a set of matrices. When $\rho(g)$ is a $d_\rho \times d_\rho$ matrix, the 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 a particular representation called the *regular representation.*

The regular representation of a group of N elements is a matrix representation of order N. We will explain the construction of the regular representation using the Dirac notation. First, we associate with each element of the group g_i a ket $|g_i\rangle$. This ket could simply be the basis state $|i\rangle$, since the elements of the group are numbered. This ensures that the kets associated with different

group elements are orthonormal by construction, $\langle g_i|g_j\rangle = \delta_{ij}$. This also ensures that the identity operator can be expressed as $\sum_{i=1}^{N} |g_i\rangle \langle g_i|$. The regular representation of g_k is then given by,

$$R(g_k) = \sum_{j=1}^{N} |g_k \cdot g_j\rangle\langle g_j| . \tag{50}$$

The matrix elements of this representation are, $R_{ij}(g_k) \equiv \langle g_i|R(g_k)|g_j\rangle = \langle g_i|g_k \cdot g_j\rangle$. From the defining properties of a group it can be easily seen that multiplying every element in the group by the same element just permutes the elements of the group. This means that $R(g_k)$ matrices are always permutation matrices and are hence unitary. We can prove that the regular representation is a representation using simple algebra,

$$\begin{aligned} R(g_k) \cdot R(g_m) &= \sum_{i=1}^{N}\sum_{j=1}^{N} |g_k \cdot g_i\rangle \langle g_i|g_m \cdot g_j\rangle \langle g_j| , \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N} |g_k \cdot g_m \cdot g_j\rangle \langle g_i|g_m \cdot g_j\rangle \langle g_j| , \\ &= \sum_{j=1}^{N} |g_k \cdot g_m \cdot g_j\rangle\langle g_j| = R(g_k \cdot g_m). \end{aligned} \tag{51}$$

Here 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. Below we give some explicit examples of regular representations.

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

$$R(a_0) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \; R(a_1) = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \; R(a_2) = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \; R(a_3) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}. \tag{52}$$

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])

$$R(s_1) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \; R(s_2) = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \; R(s_3) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \tag{53}$$

$$R(s_4) = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \; R(s_5) = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}, \; R(s_6) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{54}$$

Now we can finally explain 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: $\langle \mathbf{A} \rangle = \langle \psi | \, \mathbf{A} \, | \psi \rangle$.

Example 4A. Calculating matrix elements of the regular representation of the element a_2 from the Abelian group A_4 with respect to the state ψ_{13} which is the equal superposition of $|a_1\rangle$ and $|a_3\rangle$. In operator form we find:

$$\langle \psi_{12} | \, \mathbf{a}_2 \, | \psi_{12} \rangle = \frac{\langle a_1 | + \langle a_3 |}{\sqrt{2}} \left(\sum_{i=0}^{N-1} |a_2 \cdot a_i\rangle\langle a_i| \right) \frac{|a_1\rangle + |a_3\rangle}{\sqrt{2}} = \frac{\langle a_3|a_2 \cdot a_1\rangle\langle a_1|a_1\rangle}{2} + \frac{\langle a_1|a_2 \cdot a_3\rangle\langle a_3|a_3\rangle}{2} = 1 \, . \tag{55}$$

It is quite obvious that if a quantum computer is capable of finding expectation values of a unitary operator, it will be able to solve the problem of finding the matrix elements of the regular representation of a group element. This 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 ($|0\rangle \rightarrow |1\rangle$) and qubit swap ($|q_j q_i\rangle \rightarrow |q_i q_j\rangle$).

Up to this point, we have only talked about the regular representation. 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 is easy to construct using standard quantum circuits. However, for groups with a large number of elements, it requires matrix multiplication between large matrices. So for many applications, instead of regular representations one is interested in what are known as *irreducible representations*, which are matrix representations that cannot be decomposed into smaller representations. Or in other words, every matrix representation (including the regular representation) can be shown to be equivalent to a direct sum of irreducible representations, up to a change of basis. This lets us reduce the representation theory of finite groups into the study of irreducible representations. The importance of irreducible representations in group theory cannot be overstated. The curious reader may refer these notes by Kaski [60].

A result from group theory ensures that the direct sum of all irreducible representations (each has different dimensions d_ρ in general) where each irreducible representation appears exactly d_ρ times is a block diagonal $N \times N$ matrix (the group has N elements). The Fourier transform pair over this group representation can be introduced by decomposing each irreducible representation over the group elements and *vice versa*. Moreover, the above defined direct sum of all irreducible representations can be decomposed as a regular representation conjugated by the direct and inverse Fourier transform operators [59]. This result lets us find the the matrix elements of the irreducible representations given the ability to implement the regular representation.

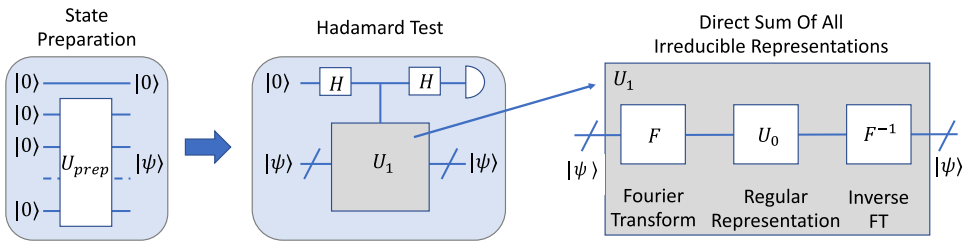


Fig. 20. Schematic diagram for the quantum algorithm

6.2 Algorithm description

In this section we will describe an algorithm to find the matrix elements of irreducible representations of a group given the ability to apply its regular representations to a quantum register in a controlled fashion. 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. 20.

Algorithm 5 Hadamard test

Input:

- The controlled unitary CU.
- Input state $|0\rangle|\psi\rangle$.

Output:

- An estimate for the real part of $\langle\psi|U|\psi\rangle$

Procedure:

Step 1. Apply H to the ancilla. This produces the state,

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}}\,|\psi\rangle$$

Step 2. Apply CU controlled on the ancilla. This produces the state,

$$\frac{|0\rangle\,|\psi\rangle + |1\rangle\,U\,|\psi\rangle}{\sqrt{2}}$$

Step 3. Apply H to the ancilla again. This gives,

$$\frac{|0\rangle\,(|\psi\rangle + U\,|\psi\rangle) + |1\rangle\,(|\psi\rangle - U\,|\psi\rangle)}{\sqrt{2}}$$

Step 4. Measure the ancillary qubit. Repeat to estimate the probability of obtaining $|0\rangle$ and $|1\rangle$.

The ancilla qubit should be prepared as $\frac{|0\rangle - i|1\rangle}{\sqrt{2}}$ to calculate the imaginary parts of the matrix element. From the pseudocode, we can see that the probability of measuring $|0\rangle$ is $P_0 = ||\frac{|\psi\rangle + U|\psi\rangle}{\sqrt{2}}||^2 = \frac{1+Re\langle\psi|U|\psi\rangle}{2}$. Hence, we find: $Re\,\langle\psi|\,U\,|\psi\rangle = 2P_0 - 1$. The reader is encouraged to work out the same steps for the imaginary part as well.

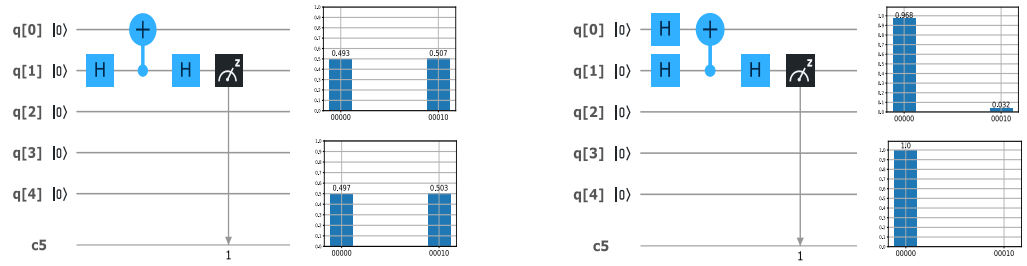


Fig. 21. Actual circuit implemented on IBM's 5-qubit computer for calculating matrix elements of the regular representation for the second element of the group S_2 and A_2 in respect to the state $|0\rangle$ on the left and $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ on the right. The expected probabilities 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.

With the Hadamard test algorithm, the problem of calculating matrix elements of an arbitrary unitary operator is reduced to the problem of effectively implementing it as a controlled gate. 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, this implementation can be done for any group as a combination of $CNOT$ and Z gates.

At the same time solutions for the direct sum of all irreducible representations U_1, which can be decomposed as $U_1(g) = F_1 U_0(g^{-1}) F_1^{-1}$, exists for any group whose 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 $S(n)$ [12], the alternating group A_n, and some Lie groups: $SU(n)$, $SO(n)$ [9], while solutions for other groups, hopefully, will be found in the future. For Abelian groups this Fourier transform implementation can be efficiently done using the QFT circuit that was discussed in the earlier sections. For non-Abelian groups the implementation is trickier and efficient implementations are not universally known.

6.3 Algorithm implemented on IBM's 5-qubit computer

The actual gate sequence that we implemented on IBM's 5-qubit computer (`ibmq_essex`) and IBM's quantum simulator to find matrix elements of the regular representation of the second element of the group S_2 is shown in Fig. 21. The matrix for this representation is simply a X gate. Hence, we have to use one CNOT gate and two Hadamard gates, plus some gates to prepare state $|\psi\rangle$ from the state $|00\rangle$. We mapped the ancilla qubit to the actual machine q_1 qubit instead of q_0, because of the machine architecture, where the first qubit can control the zero qubit but not *vice versa*. We could have used the original qubit sequence as in Fig. 20, by realizing the CNOT gate as a swapped CNOT and four Hadamard gates, but this would add more gates to the circuit and potentially more computational errors rather than just a virtual swap of the qubits.

For the irreducible representation of the same element of the group A_2 , the element is represented by the Z gate. Hence the Hadamard test requires implementing a controlled-Z gate, which is not available 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. 22. Notice that the Hadamard gate is actually the Fourier transform operator over group S_2 and A_2, while the X gate is a regular representation operator, as we mentioned earlier. Hence, such controlled-Z gate representation

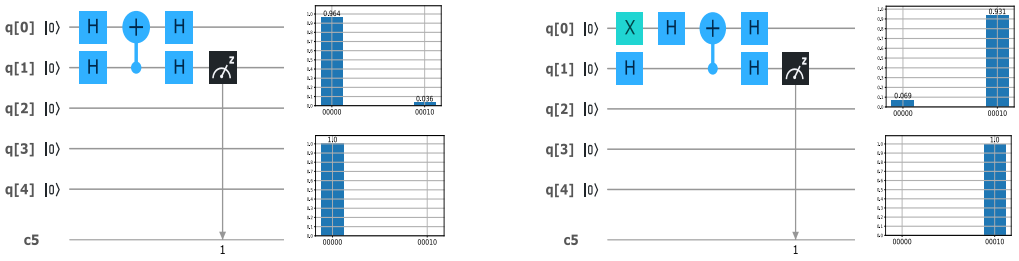


Fig. 22. Actual circuit implemented on IBM's 5-qubit computer for calculating matrix elements of the direct sum of the irreducible representations for the second element of the group S_2 and A_2 with respect to the state $|0\rangle$ on the left and $|1\rangle$ on the right. The expected probabilities 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.

is in fact the decomposition of the irreducible representation to the regular representation using Fourier transform over that group.

7 QUANTUM VERIFICATION OF MATRIX PRODUCTS

7.1 Problem definition and background

Matrix multiplication is one of the most important linear algebra subroutines. Most scientific computing algorithms use matrix multiplication in one form or another. Therefore, the computational complexity of matrix multiplication is a subject of intense study. For two $n \times n$ matrices the computational complexity of the naive matrix multiplication algorithm is $O(n^3)$ A faster algorithm for matrix multiplication implies a considerable performance improvement for a variety of computational tasks. Strassen [100] first showed that two $n \times n$ matrices can be multiplied in time $O(n^{2+\alpha})$ ($\alpha < 1$). The best known algorithm to date with $\alpha \approx 0.376$ was found by Coppersmith and Winnograd [32]. Despite that, it remains an open problem to determine the optimal 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 classical algorithm can do this with high probability in time proportional to n^2 [45].

Ref. [4] 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. [45]. Ref. [23] 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. [4].

7.2 Algorithm description

We briefly sketch the quantum algorithm from Ref. [4]. The presentation here follows from Ref. [99]. Before we discuss this algorithm we introduce the concept of *amplitude amplification.*

Many real world algorithms are probabilistic, i.e., independent runs of the algorithm on the same input will not necessarily give the same output. This is because the algorithm uses some source of randomness during its execution. Most quantum algorithms are probabilistic owing to the inherent randomness present in quantum mechanics.

Suppose that the job of our probabilistic classical/quantum algorithm is to return one of a specific set of states. Assume that we also have at our disposal an oracle that can identify the members of this set from other states. An example of this would be polynomial root finding. The set of states in this case would correspond to the roots of the polynomial. Our algorithm should return one of the roots of the polynomial and we can verify if an output is a root by plugging it in to the polynomial.

Obviously the algorithm is good only if it can return a state that is a member of this set with high probability. But how high of a success probability is good enough? For practical reasons we would like the probability of success to be a constant. That is, it should be a value independent of the problem size and other parameters in the problem. Any constant value between 0 and 1 would work here. The value $\frac{2}{3}$ is usually used in literature.

But often algorithms won't succeed with constant probability and their success provability will diminish with growing input size. In that case, how can we boost the success probability to the desired level? The classical answer to this question is to repeatedly run the algorithm until we succeed, i.e., till the algorithm outputs a state from the specific set of states that we want. If the algorithm initially had a success probability of p, after $O(\frac{1}{p})$ repetitions we are guaranteed to find the desired state with constant probability.

For quantum algorithms we can do something better. Let U be a quantum algorithm and suppose that we want this algorithm to return a state from the subspace spanned by the orthogonal states, $\{|u_i\rangle\}$. Let P be the projection operator onto this subspace, $P = \sum_i |u_i\rangle \langle u_i|$. The oracle we have is then, $O = I - 2P$. This oracle will mark the states in the desired subspace. The success probability of our algorithm is $p = \left\langle 0 \dots 0|U^\dagger PU|0 \dots 0 \right\rangle$. In this scenario we can use amplitude amplification to boost the success probability to a constant with only $O(\frac{1}{\sqrt{p}})$ repetitions. This is a quadratic speedup over the classical strategy.

Essentially, amplitude amplification is a generalization of Grover search described in Section II . In Grover search we repeatedly apply the Grover operator, $G = (2|\psi\rangle \langle\psi| - I)O$, where $|\psi\rangle$ is the uniform superposition state. Amplitude amplification uses a more general operator,

$$G_U = U(2|0\rangle \langle 0| - I)U^\dagger O. \tag{56}$$

To get the desired result we apply this to the $U|0 \dots 0\rangle$ state $O(\frac{1}{\sqrt{p}})$ times. Notice that the original Grover search is a specific case of amplitude amplification with $U = H \otimes \dots \otimes H$. In that case, the probability of getting the marked state in $|\psi\rangle$ is $\frac{1}{N}$ so we run the algorithm for $O(\sqrt{N})$ steps. The reader is referred to Ref. [21] for more details on amplitude amplification.

The matrix product verification procedure uses amplitude amplification as its outer loop. The algorithm first splits the full matrix verification problem into smaller matrix verification problems. Then it uses amplitude amplification to search if one of these verifications fail. Each of these smaller verification steps also use a Grover search to look for disagreements. So the complete algorithm uses one quantum search routine nested inside another quantum search routine. This is a common strategy used while designing quantum algorithms to improve query complexity. The full algorithm is sketched below.

The number of qubits and the circuit depth required for this algorithm is too large for it to be successfully implemented on the IBM machines. But at the heart of this algorithm is the Grover search procedure, which we have already discussed and implemented in Section II

Algorithm 6 Matrix product verification [4] [99]

Input:

• $n \times n$ matrices A, B, C.

Output:

• Verifies if $AB = C$

Procedure:

Step 1. Partition B and C into $\sqrt{n}$ submatrices of size $n \times \sqrt{n}$. Call these B_i and C_i respectively. $AB = C$ if and only if $AB_i = C_i$ for all i.

Step 2. Use amplitude amplification over i on these steps:

Step 2a. Choose a random vector x of dimension $\sqrt{n}$.

Step 2b. Compute $y = B_i x$ and $z = C_i x$ classically

Step 2c. Verify equation $Ay = z$ by Grover search. Search for a row j such that $(Ay - z)_j \neq 0$

8 GROUP ISOMORPHISM

8.1 Problem definition and background

The *group isomorphism* problem, originally identified by Max Dehn in 1911 [35], 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). At the end of Section 5 we saw an example of two isomorphic groups, S_2 and A_2. These two are the same group in terms of how the group operation works on the group elements, but are defined in different ways. More precisely, two groups, $(G_1, \cdot)$ and $(G_2, *)$ are called isomorphic if there is a bijection, $f : G_1 \rightarrow G_2$, between them such that, $f(g_1 \cdot g_2) = f(g_1) * f(g_2)$.

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 $U_a : |y\rangle \rightarrow |ay\rangle$, for any $a \in G$.

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]. As we saw before, abelian simply means that the operation $(\cdot)$ used to define the group is commutative, such that $a \cdot b = b \cdot a$, for $a, b \in G$. Although Shor's approach is specifically intended to leverage a quantum period-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 [25] 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.

8.2 Algorithm description

The procedure presented in Algorithm 7 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.

Since the procedure in Algorithm 7 is mostly classical, we shall treat the task of finding the generators of the hidden subgroup in **Step 1** as the most critical for us to explore. This task is commonly referred to as 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

Algorithm 7 Decompose($a_1, \ldots, a_k, q$), of Cheung and Mosca [25]

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 $g : \mathbb{Z}_q^k \rightarrow G$ by mapping $(x_1, \ldots, x_k) \rightarrow g(x) = a_1^{x_1} \cdots a_k^{x_k}$.
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 $\{g(y_1), \ldots, g(y_l)\}$.

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 7 is described in algorithm 8.

Algorithm 8 Solution to the hidden subgroup problem (for finite abelian groups). Based on Ref. [77]

Input:
- Two quantum registers.
- Elements of the finite abelian group A (or the generating set).
- A function g, such that $g : A \rightarrow X$, 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 $g(a)$.

$$\rightarrow \frac{1}{\sqrt{|A|}} \sum_{a \in A} |a\rangle |g(a)\rangle \tag{57}$$

Step 4. Apply inverse Fourier transform.

$$\rightarrow \frac{1}{\sqrt{|A|}} \sum_{l=0}^{|A|-1} e^{2\pi i l a/|A|} |\hat{g}(l)\rangle \tag{58}$$

Step 5. Measure the phase from first register.

$$\rightarrow l/|A| \tag{59}$$

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

Like the period-finding approach used in quantum factorization in Section V, Algorithm 8 is heavily based on the concept of phase estimation. Note that the Fourier transform in Eq. 58 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

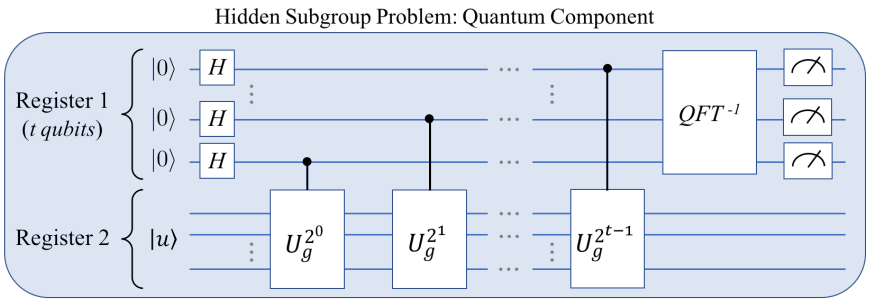


Fig. 23. Basic phase-estimation quantum circuit needed to solve the general hidden subgroup problem in algorithm 8. Here, $|u\rangle$ is an eigenstate of the unitary operator U.

$$|K| = \sum_{h \in K} e^{-2\pi i l h/|A|}, \tag{60}$$

and that knowledge of l can be used to determine both the elements and generating set of K. As discussed by Nielsen and Chuang [77], the final step in algorithm 8 can be accomplished by expressing the phase as

$$\rightarrow e^{2\pi i l a/|A|} = \prod_{i=1}^{M} e^{2\pi i l_i a_i/p_i}. \tag{61}$$

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. 23. This simplified circuit includes steps 1-5 of algorithm 8, and makes it clear that all forms of the HSP (order-finding, period-finding, discrete logarithm, etc.) are extensions of quantum phase estimation.

8.3 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. 23. We also chose a specific instance of the HSP: the problem of finding the period of a mod n. In Fig. 24, 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 mod 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 mod 15 operations, one can simply used the circuits developed by Markov and Saeedi [87]. The code in Fig. 24 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. 24 can be directly implemented using the IBM Qiskit interface, the resulting QASM code is not expected to lead to accurate results on the IBMX4 (or IBMX5). This is because the generated circuit is long enough for decoherence error and noise to ultimately dominate the measured state. In other words, the physical hardware requires further

```
#================================================================#
#----------  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. 24. Simple implementation of the quantum period-finding algorithm in Qiskit

optimization to reduce the number of gates used between the initial state preparation and the final measurement.

9 QUANTUM PERSISTENT HOMOLOGY

9.1 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

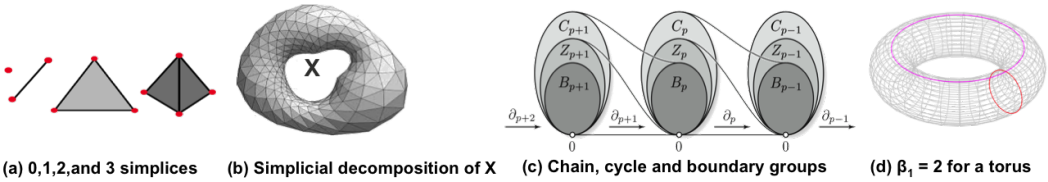


Fig. 25. Examples of simplices, simplicial decomposition of a topological space X, relationships among groups C_p (chains), Z_p (cycles), and B_p (boundaries) under the action of boundary homomorphisms ∂_p, and the example of the torus.

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 [68] is the focus of this section.

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 [68]. 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, $B_k \subseteqq Z_k$. The relationships between chain groups, cycles and boundaries as established by the boundary homomorphisms is illustrated in Fig. 25(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 [68] diagonalizes the Laplacian to compute Betti numbers.

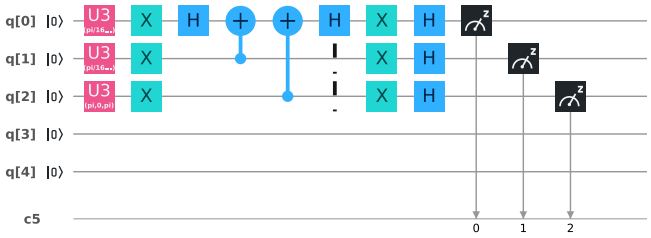


Fig. 26. Grover's Algorithm circuit implemented on the 5 qubit quantum computer showing 3 qubits being used with the multiple solution version of Grover's Algorithm. U3 gates are used to input the scaled distances between points.

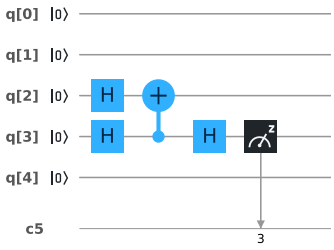


Fig. 27. Quantum Phase Estimation Algorithm circuit implemented on the 5 qubit quantum computer showing how the quantum density matrix implemented on qubits 0, 1, and 2 would be applied to obtain a classical measurement on qubit 3.

9.2 Quantum algorithm description

A quantum algorithm for calculating Betti Numbers is presented in [68]. The algorithm uses Grover's search combined with phase estimation to find the dimension $Ker(\Delta_k)$. Grover's algorithm is used to prepare a suitable initial state for the phase estimation. We will demonstrate how phase estimation can be used to estimate the dimension of the kernel of an eigenspace. Suppose that we have an $N \times N$ unitary operator $U = e^{i2\pi H}$ on which we will apply phase estimation. Let $\left|u_j\right\rangle$ and $e^{i2\pi\lambda_j}$ be the eigenvectors and eigenvalues of U. Now given a starting state $\frac{1}{\sqrt{N}}\sum_{j=1}^{N}\left|u_j\right\rangle$ we know that the phase estimation subroutine will effect the following transformation,

$$\frac{1}{\sqrt{N}}\sum_{j=1}^{N}\left|u_j\right\rangle|0\rangle \rightarrow \frac{1}{\sqrt{N}}\sum_{j=1}^{N}\left|u_j\right\rangle\left|\tilde{\lambda}_j\right\rangle, \tag{62}$$

where $\tilde{\lambda}_j$ are approximations to the original eigenphases. Now consider a measurement on the ancillary register that stores these eigenphases. If one of the λ_j were zero, we can see that the probability of measuring $|0\rangle$ on the second register is equal to $\frac{dim(Ker(H))}{N}$. Moreover the probability of measuring any $\left|\tilde{\lambda}_j\right\rangle$ will similarly be related to the dimension of its eigenspace. So by estimating these probabilities, we can figure out the dimensions of the eigenspaces. Notice that the performance and correctness of the procedure will depend on the precision of $\tilde{\lambda}_j$. This will in turn depend on the number of ancillary qubits used in the procedure. For this procedure to work it was crucial

that we started with the uniform superposition of all the eigenstates. This is a correct but naive way to go about the problem especially if the dimension of the null space is exponentially small compared to the size of the matrix. But, this technique will work equally well if the input to the procedure was a classical mixture of some of the eigenstates such that the probability of the null states in the said mixture was related to the dimension of the null space. This would let us recover the dimension of the null space from the measurement probabilities. The algorithm to find Betti numbers uses such a generalization of the naive procedure illustrated above.

We will roughly sketch the full algorithm without going into the details. We associate with each simplex σ_k a computational basis state $|\sigma_k\rangle$ such that the 1s in $|\sigma_k\rangle$ correspond to the points chosen in σ_k. Then the algorithm roughly goes as follows:

(1) First we construct a uniform superposition over all the simplex states in a set denoted by S_k^ϵ. The precise definition of the set is not important to us now. The crucial fact is that there exists an easily computable function that determines the membership of a simplex to this set. This requires an additional input of scaled distance between points. This function can then be used as an oracle in Grover search to compute the desired state.
(2) Then this state is modified to produce an equiprobable mixture over all the simplicies in S_k^ϵ. This mixture is represented as a quantum density matrix ρ. Readers unfamiliar with the concept of density matrices should read the section on quantum tomography.
(3) Now perform phase estimation on ρ to find 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. In the implementation of Grover's Algorithm is shown in Fig. 26, only one iteration of Grover's algorithm is shown. 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. The Quantum Experience 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, phase estimation 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 phase estimation quantum circuit for this purpose is shown in Fig. 27. Hence the main bottleneck here is seen to the the coherence time of the computer.

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. 28 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. 29 showing the decoherence rates with quantum composer timesteps. The coherence rates here measure the 'quantumness' of the qubits as detailed in Ref. [105]. Note that although the coherence rates are fairly high for individual qubits, the overall qubit coherence percentages are far less. This

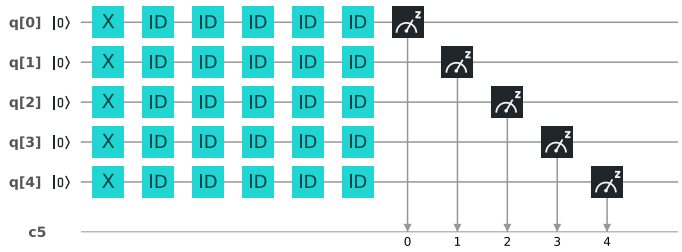


Fig. 28. The quantum algorithm applied to estimate the Decoherence time of the 5 qubit quantum computer as implemented in the Quantum Composer to measure the decoherence after 5 timesteps. Note that this produces an optimistic estimate as the "id" quantum gates utilize minimal time whereas other gates, such as CNOT and U3 gates, could use more time.

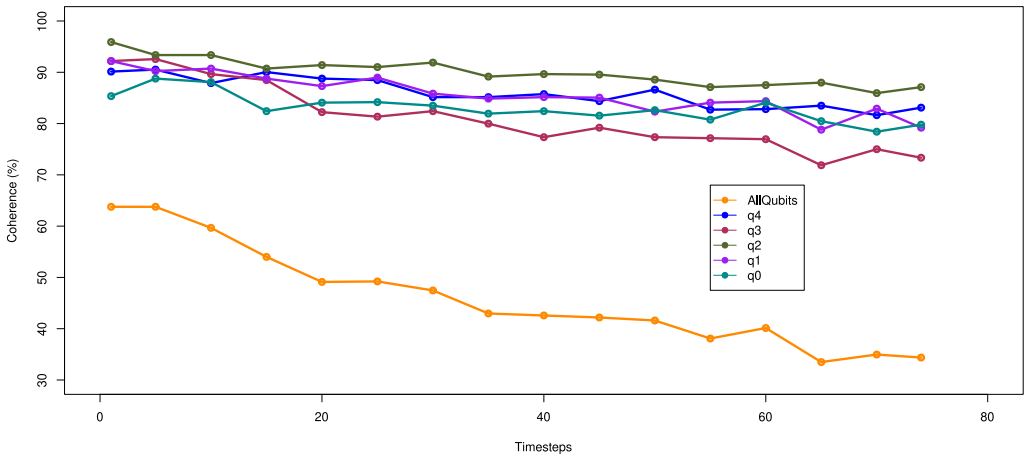


Fig. 29. Timesteps available with the Quantum Composer are shown on the x axis, ranging up to 74. Coherence percentage is calculated for all individual qubits and is plotted on the y axis. Coherence percentages of all 5 qubits combined is also shown on the y axis. Note that the coherence percentage rate falls below 50 percent between the 15th and the 20th timesteps.

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 in an actual machine. It is also interesting that qubit 3 seems to decohere at a faster rate than the other qubits past timestep 15.

10 QUANTUM RANDOM WALKS

10.1 Problem definition and background

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 [52] with an oracle constructed from the adjacency matrix.

But for some problems Grover's algorithm is insufficient to achieve optimal query complexity. In such cases, a quantum random walk can sometimes be useful in reducing the query complexity of an algorithm further. An example of this is the quantum algorithm for element distinctness by Ambainis [5]. Additionally, quantum walk algorithms can also be used to search and find graph properties [29, 38, 42, 61, 62, 71]. Quantum random walks can be seen as a quantum mechanical generalization of classical random walks. Quantum random walk algorithms come in two forms, discrete time quantum walks and continuous time quantum walks [61]. 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 expressed as a Hamiltonian, whose time evolution is then simulated. Quantum random walks can be used to walk a graph [38, 62], search for marked vertices [42], and to solve s-t connectivity [62]. An excellent survey of this approach to quantum search can be found in Ref. [88].

Most quantum algorithms that solve graph problems requires an oracle that knows the properties of the underlying graph. A graph properties oracle can be assembled as a circuit based on the adjacency matrix of the graph and linear algebra transformations. For example, a quantum circuit for finding maximal cliques in a graph with n nodes, requires an oracle workspace of n^2 data qubits and n^2 ancilla qubits (see [109]). Each oracle call requires execution of $6n^2$ Toffoli gates and $2n$ CNOT gates. 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 [28] typically use the adjacency matrix to create the oracles. Here we will not get into using quantum random walks to solve such problems. Instead we will demonstrate how to implement a simple quantum random walk on a quantum computer.

10.2 Example of a quantum random walk

Quantum random walks or simply quantum walks are quantum analogues of classical random walks and Markov chains. Unlike the continuous time quantum walk, the discrete time quantum walk algorithm requires the use of one or more coin qubits representing the number of movement choices from each graph vertex. These extra coin degrees of freedom are necessary to ensure unitarity of the quantum walk. An appropriate unitary transformation on these coin qubits then acts like the quantum version of a random coin toss, to decide the next vertex for the walker.

Intuitively, the quantum walk is very similar to its classical cousin. In a classical walk, the walker observes some random process, say a coin toss, and decides on his next step conditioned on the output of this random process. So for a classical random walk, the walker is given a probability to make a transition. In a quantum walk, on the other hand, the random process is replaced by a quantum process. This quantum process is the application of the coin operator, which is a unitary matrix. So the next step of the walker is controlled by a complex amplitude rather than a probability. This generalization, from positive numbers to complex numbers, makes quantum walks more powerful than classical random walks.

The full Hilbert space for the discrete quantum walk on a cycle with $N = 2^n$ nodes can then be constructed as follows. We use an n qubit register to represent the nodes of the graph as bit strings. For the cycle every node has only two neighbours, so the coin space only needs a dimension of 2. Hence, only one extra coin qubit is required. The basis vectors of the coin ($|0\rangle$ and $|1\rangle$) will denote the right and left neighbours. So a basis state in the full Hilbert space will have the form $|k, q\rangle$, where k is represents a node on the cycle and q is a single bit representing the coin state.

The quantum walk is then a product of two operators, the shift operator (S) and the coin operator (C). As we mentioned before the coin operator only acts on the coin qubit. The coin operator can be

in principle any unitary that mixes the coin states, but here we will use the Hadamard coin which is just the H gate on the coin qubit,

$$C\,|k,q\rangle = I \otimes H\,|k,q\rangle = \frac{|k,0\rangle + (-1)^q\,|k,1\rangle}{\sqrt{2}}. \tag{63}$$

The shift operator acts on both the registers. It moves the walker to the left or right depending on the coin state and then flips the coin state,

$$S\,|k,q\rangle = |k + (-1)^q, q \oplus 1\rangle \tag{64}$$

The quantum walk then proceeds by applying these two operators in alternation. A p step quantum walk is just the operator $(SC)^p$. This type of a walk was first introduced in Ref. [92] and is sometimes referred to as a 'flip-flop' quantum walk.

The definition of these operators can change for different types of quantum walk. The coin operator can be a Hadamard gate or a sub-circuit that results in mixing the coin states. 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 state of the coin. A simple pseudo-code for implementing the quantum walk is given in Algorithm 9.

Algorithm 9 Discrete time quantum walk

Input:

- Two quantum registers. The coin register and the position register.
- Number of steps, T.

Output:

- State of the quantum walk after T steps.

Procedure:

Step 1. Create the initial state. The initial state depends on the application. For instance, in quantum search algorithms, the initial state is the uniform superposition state.

for $0 \leq k < T$ **do**

Step 2a. Apply the coin operator, C, to the coin register.

Step 2b. Apply the shift operator, S. This shifts the position of the walker controlled on the coin state.

end for

Step 3. (Optional) Measure the final state.

10.3 Algorithm implementation using Qiskit on IBM Q

In this section we will implement a simple quantum walk on Qiskit and execute it on both the simulator and `ibmq_vigo`, which is a 5 qubit machine available on IBM Q. We will test the quantum walk on a simple 4 vertex cycle with the vertices labels as given in Fig. 30.

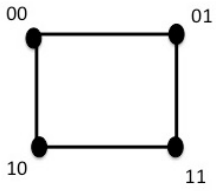


Fig. 30. A graph of 4 nodes in the form of a square is used for the random walk algorithm. The starting vertex is labeled 00. The next possible vertex choices vary by 1-bit in their labels, 01 and 10. The quantum walk algorithm will walk around the graph.

The coin operator in Eq. (63) is just the H gate acting on the coin qubit. The shift operator defined in Eq. (64) is more non-trivial. We can implement it by the circuit given in Fig. 31.

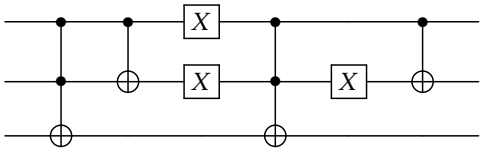


Fig. 31. Quantum circuit for the shift operation on the 4 vertex cycle. The top qubit is the coin qubit.

Running the walk for multiple steps requires us to apply the shift operator circuit many times. So it would be tedious to implement the quantum walk on the IBM Q graphical interface. Instead we can use Qiskit to design the shift operator as a user defined gate and then run the walk for multiple steps using a simple `for` loop. The Qiskit code for this is given in Fig. 32.

We ran this Qiskit code for 4 steps of the quantum walk. We chose 4 steps since, a simple calculation shows that, starting from $|000\rangle$ and applying $(SC)^4$ will concentrate all the probability to the state $|100\rangle$. This is confirmed by running the Qiskit code on the simulator. But running the same code on `ibm_vigo` gave $|100\rangle$ with only 21.7% probability. The rest of the probability was distributed among the other basis states, but $|100\rangle$ was still the state with the largest probability. This poor performance is due to the circuit having large depth. We can expect to get better results by running the quantum walk for a single step. After a single step, starting from $|000\rangle$, the state of the system is $\frac{|111\rangle+|010\rangle}{\sqrt{2}}$. This is again confirmed by the simulator. Running on `ibm_vigo`, we got $|111\rangle$ with 33.5% probability and $|010\rangle$ with 28.5% probability.

11 QUANTUM MINIMAL SPANNING TREE

11.1 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? Fig 33 shows a model of a simple road network as a graph, together with a minimal spanning tree.

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

```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit import Aer, execute

n_steps =  4          #Number of steps

#Defining the shift gate

shift_q =  QuantumRegister(3)                                  #3 qubit register
shift_circ = QuantumCircuit (shift_q, name='shift_circ')       #Circuit for shift operator
shift_circ.ccx (shift_q[0],  shift_q[1], shift_q[2])           #Toffoli gate
shift_circ.cx ( shift_q[0], shift_q[1] )                       #CNOT gate
shift_circ.x ( shift_q[0] )
shift_circ.x ( shift_q[1] )
shift_circ.ccx (shift_q[0],  shift_q[1], shift_q[2])
shift_circ.x ( shift_q[1] )
shift_circ.cx ( shift_q[0], shift_q[1] )

shift_gate = shift_circ.to_instruction()          #Convert the circuit to a gate

q = QuantumRegister (3, name='q')                 #3 qubit register
c = ClassicalRegister (3, name='c')               #3 bit classical register
circ = QuantumCircuit (q,c)                       #Main circuit
for i in range(n_steps):
    circ.h (q[0])                                 #Coin step
    circ.append (shift_gate, [q[0],q[1],q[2]])    #Shift step

circ.measure ([q[0],q[1],q[2]], [c[0],c[1],c[2]])
```

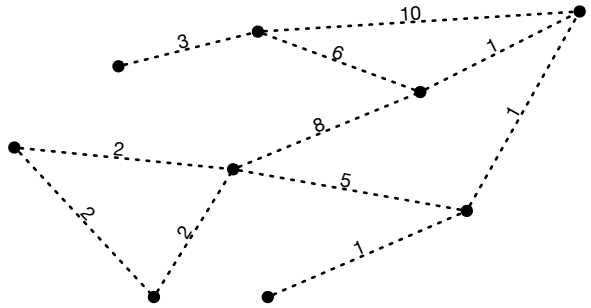
Fig. 32. Qiskit code to implement the quantum walk on a 4 vertex cycle.

(i.e., there is exactly one path between any two vertices). It is not hard 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

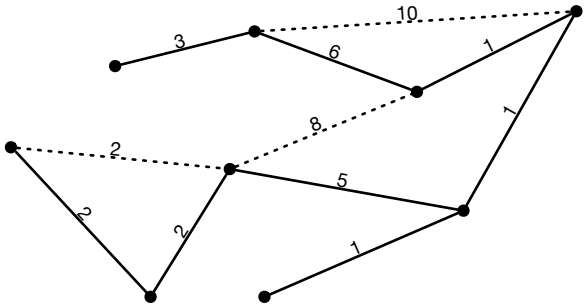
$$\sum_{e \in E_T} w(e). \tag{65}$$

11.2 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 [39]. 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 $O(m \log n)$. In particular, this is the time complexity of Borůvka's algorithm [19]. The quantum algorithm combines Borůvka's algorithm together with the quantum search algorithm of Grover [52].



(a) The weighted graph model.



(b) A minimal spanning tree.

Fig. 33. A graph modeling repair costs of a simple transportation network (a) together with (b) its minimal spanning tree (the solid edges). The sum of the weights of the edges in the minimal spanning tree is 21.

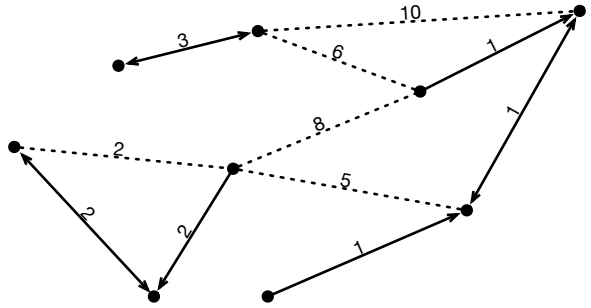


Fig. 34. The first two steps of Borůvka's algorithm. Starting with each node as a distinct tree, find the minimal weighed edge between each tree and the rest of the trees. The direction of the solid edges indicates the edge is the minimal weighted edge for the source node. The components connected by solid edges (disregarding the directions) will form the trees at the start of the second run of step (2) of Borůvka's algorithm

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 Fig 34. Formally, we have

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

(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).

In the previous sections we used Grover search to look for a single item in a list of N elements. But the search algorithm will work even if there are M elements in the list that are marked by the oracle. One of these marked elements can then be found using $O(\sqrt{\frac{N}{M}})$ queries to the oracle.

In the algorithm above, we need to find the minimal element of an appropriate list. Clearly this can not be implemented directly as an oracle without actually inspecting each of the list elements. Luckily, there is a simple work around given by Durr et al [39] which involves multiple calls to the Grover algorithm as described in Algorithm 10.

Algorithm 10 Minima finding algorithm

Input:

- A unitary implementation a function F on a list of N elements,

$$U_F |x\rangle |y\rangle = |x\rangle |y \oplus F(x)\rangle .$$

Output:

- $|x^*\rangle$ such that $F(x^*)$ is the minimum of the function over the list.

Procedure:

Step 1. Pick a random j from the list.

for $0 \leq k < T$ **do**

Step 2a. Do Grover search [20] with the oracle for function f_j such that,

$$f_j(i) = \begin{cases} 1 & \text{if } F(i) \leq F(j) \\ 0 & \text{if } F(i) > F(j) \end{cases}$$

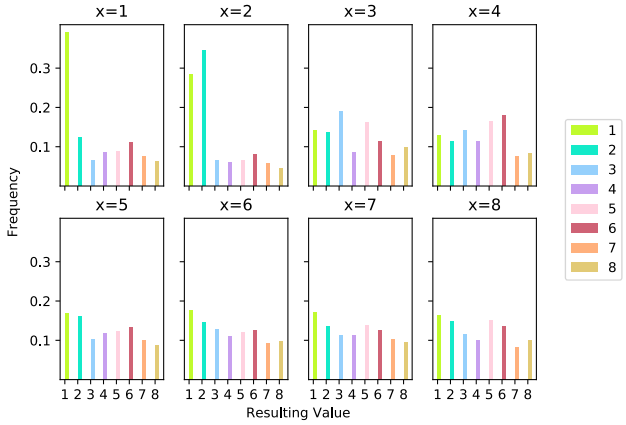
Step 2b. Update j with the result of Grover search.

end for

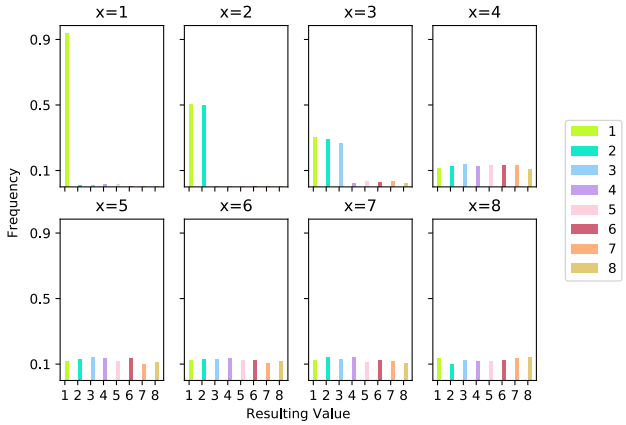
A probabilistic analysis shows that $T = 22.5\sqrt{N} + 1.4 \log_2^2(N)$ suffices to find the minimal element with high probability [40] . The inner loop of the algorithm uses a Grover search routine with potentially multiple marked items. But the number of marked items is not known beforehand. This poses problem as Grover search being a unitary algorithm needs to be stopped exactly at the right number of iterations to give the correct answer. Running the procedure for longer deteriorates the quality of the answer. If the number of marked items is unknown the stopping criterion of the algorithm is also unknown. This problem can be rectified using some extra steps by a technique given in Boyer et al [20]. We have to use this modified version of Grover search in the inner loop.

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 two 3-bit numbers. Therefore we implemented the minima finding algorithm by hard coding the oracle for each of eight possible functions f_x: $\{f_x(i) = 1 \text{ if } F(i) \leq F(x)\}$. The results are shown in Figure 35. The QASM code for implementing $f_2(i) = 1$ if $F(i) \leq F(2)$ required just under 100 lines of code (more than 100 individual gates.) The results, even when using the simulator are not good when $k \geq N/4$ elements are marked. 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_x, which however requires an extra qubit.



(a) IBM Q Implementation



(b) Simulator Implementation

Fig. 35. The results of running 1000 trials of the minima finding algorithm on both (a) the `ibmqx4` chip and (b) the IBM simulator to find values less than or equal to the input x.

12 QUANTUM MAXIMUM FLOW ANALYSIS

12.1 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

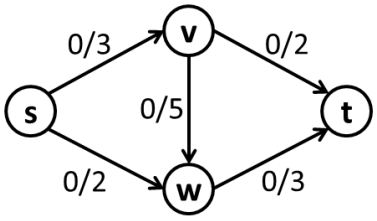


Fig. 36. A simple directed graph representing flows and capacities. Conventionally, the state of the flow problem is indicated by the current flow relative to the capacity on any directed link using the notation f/c.

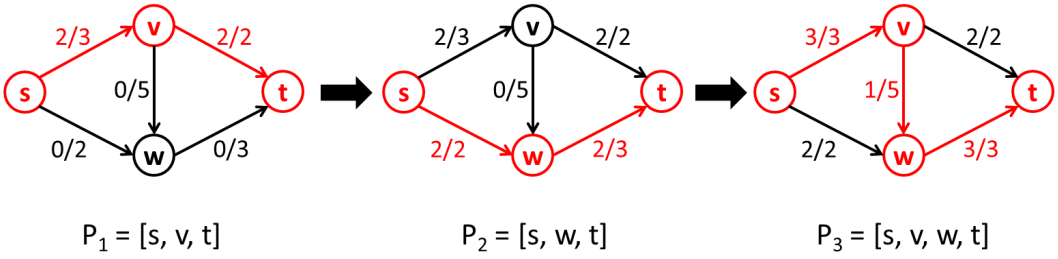


Fig. 37. The Ford-Fulkerson solution to the max-flow problem in three steps. Each step represents the application of an augmenting path to the previous flow state.

as finding a minimum cut set, and finding a maximal graph matching. The Ford-Fulkerson algorithm [44] 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 [6] claims complexity $O(n^{7/6}\sqrt{m})$.

The best classical implementations of the max-flow solver involve several important improvements [41], 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 36 and 37. 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 by f/c, with the residual capacity implicitly given by $c - f$. In Figure 36, the initial flow has been set to zero.

The basic steps in the solution to the max-flow problem are illustrated by Figure 37. 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 augmenting that path blocks the remaining paths, flows would have to be reversed before the algorithm could proceed.

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.

12.2 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 [20]. 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. Like in Section XI, we will be oblivious to the number of marked items Grover's algorithm is searching for. So once again we have to use techniques from Ref.[20] while performing the search.

Here we will describe how to build a layered graph partition. In a layered graph partition each vertex in the graph is assigned to thew i-th layer such that edges of the graph only connect between i-th and $(i+1)$-th layers. 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 described in Algorithm 11.

Algorithm 11 Layered graph partitioning

Input:
- Adjacency information of the graph (Adjacency matrix, list of edges,etc.)
- Source vertex s.

Output:
- $\mathcal{L}$ such that $\mathcal{L}[i]$ is the layer number of the i-th vertex.

Procedure:

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

Notice that the oracle for Grover search required for this algorithm is one that marks all the neighbours of x whose layer number is currently set to ∞. Grover's search speeds up the layers assignment of the vertices by quickly finding all the entries in the layer array $\mathcal{L}$ 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 without a priori knowing the number of such values. The size of a circuit required to do a full layered graph partitioning makes it impractical to implement it on the IBM machine. But the heart of the algorithm is Grover search, which we have already implemented earlier.

13 QUANTUM APPROXIMATE OPTIMIZATION ALGORITHM

13.1 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 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,

$$\begin{aligned} &\text{maximize: } \sum_{\alpha=1}^{m} C_\alpha(z) \\ &z_i \in \{0,1\}\ \forall i \in \{1,\ \ldots, n\} \end{aligned} \tag{66}$$

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 [64], and is an optimization variant of the well-known satisfiability problem, which is NP-Complete [31]. Hence, solving an instance of Eq. (66) in practice can be computationally challenging.

To provide a concrete example of Eq. (66), let us consider the MaxCut problem. As input, the MaxCut problem takes a graph $G = (V, E)$, which is characterized by a set of nods V and a set of undirected edges E. The task is to partition the nodes into two sets, such that the number of edges crossing these sets is maximized. Figure 38 provides an illustrative example, in which a graph with five nodes and six edges is partitioned into two sets that result in a cut of size five.

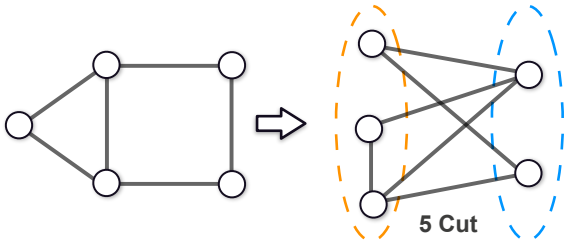


Fig. 38. An illustration of the MaxCut problem.

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

$$\begin{aligned} &\text{maximize: } \sum_{i,j \in E} \frac{1}{2}(1 - \sigma_i \sigma_j) \\ &\sigma_i \in \{-1, 1\}\ \forall i \in N \end{aligned} \tag{67}$$

In this formulation, there is one binary decision variable for each node in the graph, indicating which set it belongs to. 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 (67) will be a maximal cut of the graph G. Interestingly, the form of Eq. (67) also highlights that finding a maximal cut of G is equivalent to finding a ground state of the antiferromagnet of G in an Ising model interpretation. In either case, it is clear that Eq. (67) conforms to the structure of Eq. (66). Note that the linear transform $z = (\sigma + 1)/2$ can be used to convert the variables from the $\sigma \in \{-1, 1\}$ space to the $z \in \{0, 1\}$ space.

13.2 Algorithm description

The Quantum Approximate Optimization Algorithm (QAOA) as proposed in [43] leverages gate-based quantum computing for finding high-quality solutions to combinatorial optimization problems that have the form of Eq. (66). To apply this algorithm the user first translates the clause functions $C_\alpha(z)$ into equivalent quantum clause Hamiltonians $\mathbf{C}_\alpha$ and then selects a number of rounds $r \geq 1$ and two angles per round, $0 \leq \beta[k] \leq \pi$ and $0 \leq \gamma[k] \leq 2\pi$ for the k-th round. The pseudocode for QAOA is given in Algorithm 12.

Algorithm 12 Quantum approximate optimization algorithm

Input:

- Number of rounds of optimization r
- Two size r array of angles, γ and β.
- Hamiltonians $\mathbf{C}_\alpha$ corresponding to the clauses of the optimization problem.

Output:

- An approximation to the solution of problem in Eq. (66).

Procedure:

Step 1. Construct the n-qubit uniform superposition state by applying $H^{\otimes n}$ to $|0 \ldots 0\rangle$

for $1 \leq k \leq r$ **do**

Step 2a. Apply $\prod_{\alpha=1}^{m} e^{-i\gamma[k]\mathbf{C}_\alpha}$

Step 2b. Apply $\prod_{j=1}^{n} e^{-i\beta[k]X_j}$

end for

Step 3. We will call the state so constructed $|\beta, \gamma\rangle$. The expectation value, $\sum_{\alpha=1}^{m} \langle\beta, \gamma|\mathbf{C}_\alpha|\beta, \gamma\rangle$, gives an approximate solution to the problem.

For an appropriate selection of r, β, γ, this algorithm will give a high-quality solution to Eq. (66). As the number of rounds used increases, the quality of the solution produced by the above algorithm also increases provided that the angles chosen for the previous rounds are optimal. Conducting an exhaustive search over a fine grid is proposed in [43] for the selection of each round's optimal β, γ angles. The use of a quantum-variational-eigensolver is also possible [74, 79].

The translation of the clauses to Hamiltonians and the determination of the final expectation value depends on the specific optimization problem being solved. To provide a concrete example of the generic QAOA formulation, let us consider its application to the MaxCut problem give in Eq. (67). 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 clause Hamiltonian $\mathbf{C}_\alpha$ needs to be designed; (2) Each clause only involves two decision variables, which keeps the structure of $\mathbf{C}_\alpha$ relativity simple. The design of MaxCut clause

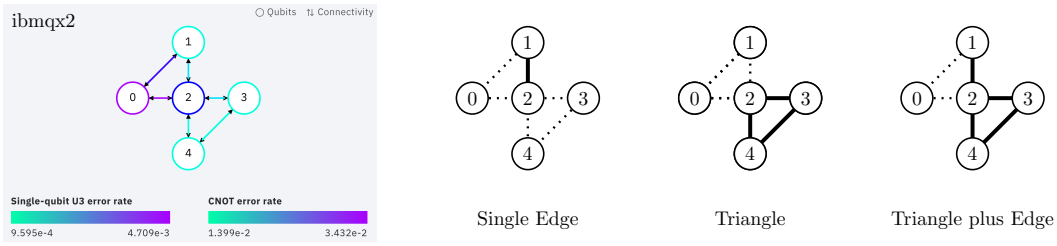


Fig. 39. The CNOT connectivity and error rates of the ibmqx2 Computer (left) followed by the Single Edge (center left), Triangle (center right) and Four edge (right) graphs considered in the proof-of-concept experiments.

Hamiltonian is as follows,

$$C(i,j) = \frac{1}{2}(1 - \sigma_i\sigma_j) \tag{68}$$

$$\begin{array}{cc} \sigma_i = -1 & \sigma_i = 1 \\ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} & \begin{array}{l} \sigma_j = -1 \\ \sigma_j = 1 \end{array} \end{array} \tag{69}$$

$$\begin{array}{c} \begin{array}{cccc} |00\rangle & |01\rangle & |10\rangle & |11\rangle \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{array}{c} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{array} \end{array} \tag{70}$$

$$C_{i,j} = \frac{1}{2}(I - Z_i \otimes Z_j) \tag{71}$$

Eq. (68) presents the binary function used by each edge in the objective of Eq. (67). Eq. (69) shows the enumeration of all inputs and outputs of the binary function, and Eq. (70) illustrates how to encode these inputs and outputs into a quantum Hamiltonian. Finally, the quantum Hamiltonian in Eq. (70) can be compactly written as in Eq. (71).

These clause Hamiltonians can then be used in the QAOA algorithm. Notice that the clause Hamiltonians here are all combinations of Z gates. This makes finding the final expectation value very simple. For each run of the algorithm, one only needs to measure the final state in the computational basis. This measurement will give a bit string that corresponds to an assignment to the classical σ_i variables. The output of the algorithm is then found by estimating the expectation value of $\sum_{i,j\in E} \frac{1}{2}(1 - \sigma_i\sigma_j)$ over independent runs of the algorithm.

13.3 QAOA MaxCut on ibmqx2

This section investigates the implementation of the QAOA MaxCut algorithm on the ibmqx2 quantum computer (Figure 39). The first challenge is to transform the QAOA algorithm from its mathematical form into a sequence of operations that are available in the IBM Quantum Experience platform. For the sake of convenience we will mention here the gates we will use in the ensuing

discussion,

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

The inner loop of the algorithm first requires the application of the γ angle with the clause Hamiltonians. For the MaxCut Hamiltonian, this can be expanded as follows,

$$e^{-i\frac{\gamma[k]}{2}(I-Z_a\otimes Z_b)} = e^{-i\gamma[k]\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-i\gamma[k]} & 0 & 0 \\ 0 & 0 & e^{-i\gamma[k]} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{72}$$

After some derivation, it is observed that this gate can be implemented as a combination of two CNOT gates and one $U_1(-\gamma)$ gate, as indicated in Figure 40. It is also interesting to note the alternate implementation of this gate in [30], which leverages a different variety of gate operations [96].

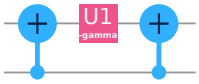


Fig. 40. An IBM Quantum Experience score illustrating an implementation of the MaxCut edge gate (72).

The next term in the loop is the application of the β angle, which is expanded as follows,

$$e^{-i\beta[k]X} = e^{-i\beta[k]\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}} = \begin{pmatrix} \cos(\beta[k]) & -i\sin(\beta[k]) \\ i\sin(\beta[k]) & \cos(\beta[k]) \end{pmatrix} \tag{73}$$

Careful inspection of the IBM Quantum Experience gates reveals that this operation is implemented by $U_3(2\beta_k, -\pi/2, \pi/2)$. So we need to apply this gate to every qubit in the register.

Putting all of these components together, Figure 41 presents an IBM Quantum Experience circuit for implementing QAOA for MaxCut on the "Triangle plus Edge" graph from Figure 39 using the following parameters,

$$r = 2\colon \gamma_1 = 0.2 \cdot \pi = 0.628.., \quad \beta_1 = 0.15 \cdot \pi = 0.471..,$$
$$\gamma_2 = 0.4 \cdot \pi = 1.256.., \quad \beta_2 = 0.05 \cdot \pi = 0.157...$$

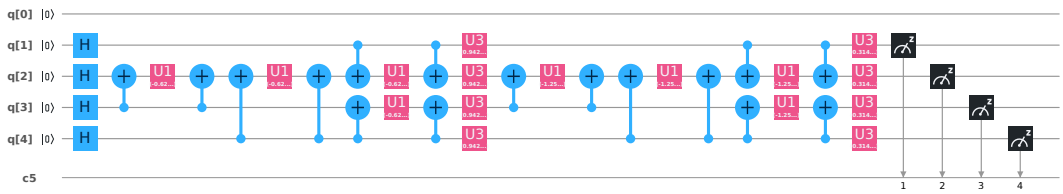


Fig. 41. An IBM Quantum Experience circuit for QAOA MaxCut with two rounds on a "Triangle plus Edge" graph.

13.4 A proof-of-concept experiment

With a basic implementation of QAOA for MaxCut in qiskit, a preliminary proof-of-concept study is conducted to investigate the effectiveness of QAOA for finding high-quality cuts in the a) Single Edge, b) Triangle and c) Triangle plus Edge graphs presented in Figure 39. 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 circuit via simulation on a classical computer; (3) *Hardware* executes the IBM Quantum Experience circuit on the `ibmqx2` hardware. The simulation computation serves to demonstrate the mathematical correctness of the proposed QAOA circuit. The hardware computation demonstrates the viability of the circuit 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. For each computation we give the expectation/mean of the returned solutions and the probability to sample the maximum cut. All of these computations are stochastic, therefore event probabilities on IBM Quantum Experience are computed based on 4096 independent runs of each computation.

a) The first experiment considers the Single Edge graph from Figure 39 (center left) and implements a 1-round QAOA with the parameters

$$r = 1\colon\ \gamma_1 = 0.5 \cdot \pi, \quad \beta_1 = 0.125 \cdot \pi.$$

The results are summarized in Table 4. 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 4. MaxCut QAOA with one round on a Single Edge.

	Random	Simulation	Hardware
Expected Size of a sampled cut	0.500	1.000	0.950
Probability of sampling a maximum cut	0.500	1.000	0.950

b) The second experiment considers the Triangle graph from Figure 39 (center right) with parameters

$$r = 1\colon\ \gamma_1 = 0.8 \cdot \pi, \quad \beta_1 = 0.4 \cdot \pi.$$

The results are summarized in Table 5. The simulation results indicate that the proposed circuit is mathematically sound. Even though the QAOA circuit for a Triangle is longer than the QAOA circuit for a Single Edge, the Hardware performance is better, most likely to the more favourable distribution of the cuts, also notable in Random.

Table 5. MaxCut QAOA with one round on a Triangle.

	Random	Simulation	Hardware
Expected Size of a sampled cut	1.500	1.999	1.904
Probability of sampling a maximum cut	0.750	1.000	0.952

c) The third experiment considers the Triangle plus Edge graph from Figure 39 (right). We run QAOA both in a 1-round and a 2-round scenario, implemented with the following parameters,

found through numerical grid searches with a resolution of $\pi/1000$ (1-round) and $\pi/20$, respectively (2-round):

$$r = 1\colon\ \gamma_1 = 0.208 \cdot \pi, \quad \beta_1 = 0.105 \cdot \pi. \qquad r = 2\colon\ \gamma_1 = 0.2 \cdot \pi, \quad \beta_1 = 0.15 \cdot \pi, \quad \gamma_2 = 0.4 \cdot \pi, \quad \beta_2 = 0.05 \cdot \pi.$$

The results are summarized in Table 6, the 2-round circuit is shown in Figure 41. Simulation and Hardware outperform Random both on 1-round and 2-round QAOA. However, the gains made by Simulation in 2-round over 1-round QAOA almost vanish on the Hardware. This degradation in performance is likely due to the double length in the circuit, making the experiment more susceptible to gate errors, environmental noise and qubit decoherence.

Table 6. MaxCut QAOA with several rounds on a Triangle plus Edge graph.

		1-round QAOA		2-round QAOA	
	Random	Simulation	Hardware	Simulation	Hardware
Expected Size of a sampled cut	2.000	2.720	2.519	2.874	2.570
Probability of sampling a maximum cut	0.375	0.744	0.652	0.895	0.727

14 QUANTUM PRINCIPAL COMPONENT ANALYSIS

14.1 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) [78]. 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

$$\Sigma = \begin{pmatrix} \mathbf{E}(X_1 * X_1) & \mathbf{E}(X_1 * X_2) \\ \mathbf{E}(X_2 * X_1) & \mathbf{E}(X_2 * X_2) \end{pmatrix}, \tag{74}$$

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

$$\Sigma = \begin{pmatrix} e_1 & 0 \\ 0 & e_2 \end{pmatrix}. \tag{75}$$

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:

$$X_1 = \text{number of bedrooms} = \{4, 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 5, 4, 3, 4\}$$
$$X_2 = \text{square footage} = \{3028, 1365, 2726, 2538, 1318, 1693, 1412, 1632, 2875, 3564, 4412, 4444, 4278, 3064, 3857\} \tag{76}$$

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:

$$\Sigma = \begin{pmatrix} 0.380952 & 0.573476 \\ 0.573476 & 1.29693 \end{pmatrix}, \quad e_1 = 1.57286\,, \quad e_2 = 0.105029\,. \tag{77}$$

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

14.2 Algorithm description

Before we discuss the algorithm we will provide a quick introduction to the concept of a *density matrix*. Density matrices are used to represent probabilistic mixtures of quantum states. Suppose that there is a quantum system whose state is not known, rather we know that it can be in one of M states, $|\psi_i\rangle$, each occurring with probability p_i. The state of this system is then represented by a density matrix ρ, defined as

$$\rho = \sum_{i=1}^{M} p_i \, |\psi_i\rangle \langle\psi_i| \,. \tag{78}$$

If the state of a system is known (with probability 1) to be $|\psi\rangle$, then the density matrix would just be $|\psi\rangle \langle\psi|$ and the system is said to be in a *pure state*. Otherwise, the system is said to be in a *mixed state*. So the density matrix can be seen as a generalization of the usual state representation with the extra ability to represent a probabilistic mixture of quantum states. From the definition of the density matrix it can be seen that it is a positive semi-definite matrix with unit trace. In fact, any matrix that satisfies these two properties can be interpreted as a density matrix. More details on the definition and interpretation of density matrices are given in the quantum tomography section (Section 20).

Density matrices are clearly more expressive than state vectors as state vectors can only represent pure states. But, even a system in a mixed state can be seen as a part of a larger system that is in a pure state. This process of converting a mixed state into a pure state of an enlarged system is called *purification.* A mixed state of an n qubit system can be purified by adding n more qubits and working with the $2n$ qubit system. Once purified, the joint system of $2n$ qubits will be in a pure state while the first n qubits will still be in the original mixed state. We will not discuss the transformations required to purify a state. Interested readers are referred to Ref. [77] for a complete discussion.

The quantum algorithm for performing PCA presented in Ref. [70] uses the density matrix representation. The algorithm discussed there has four main steps: (1) encode Σ in a quantum density matrix ρ (exploiting the fact that Σ is a positive semi-definite matrix), (2) prepare many copies of ρ, (3) perform the exponential SWAP operation on each copy and a target system, and (4) perform quantum phase estimation to determine the eigenvalues. For an implementation of this quantum PCA algorithm on a noisy simulator, we refer the reader to Ref. [66], which also gives a short-depth compilation of the exponential SWAP operation.

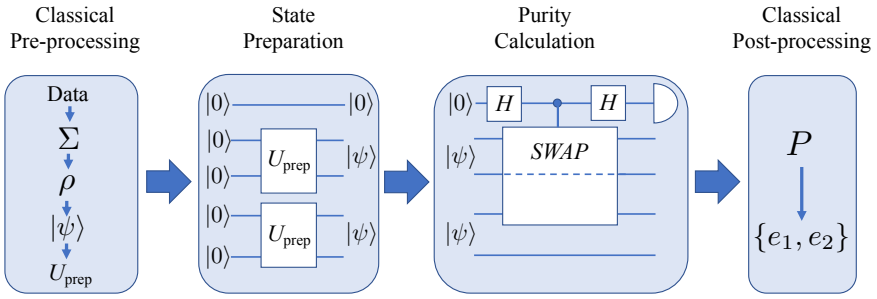


Fig. 42. Schematic diagram for the quantum algorithm for PCA, in the special case of only two features. The first step is classical pre-processing: transforming the raw data into a covariance matrix Σ, then normalizing to compute $\rho = \Sigma/\mathrm{Tr}(\Sigma)$, then purifying ρ to a two-qubit pure state $|\psi\rangle$, and finally determining the unitary U_{prep} needed to prepare $|\psi\rangle$. The second step is to prepare two copies of $|\psi\rangle$ by implementing U_{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. (79)-(80).

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 (one qubit states), 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 as 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. [57]. It is then straightforward to calculate the eigenvalues of Σ from P, as follows:

$$e_1 = \mathrm{Tr}(\Sigma) * (1 + \sqrt{1 - 2(1-P)})/2 \tag{79}$$

$$e_2 = \mathrm{Tr}(\Sigma) * (1 - \sqrt{1 - 2(1-P)})/2\,. \tag{80}$$

We remark that recently (after completion of this review article), a simpler algorithm for computing purity P was given in Ref. [27]. While the results presented in what follows use the approach in Ref. [57], the approach in Ref. [27] could lead to more accurate results.

As depicted in Fig. 42, 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, the 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, the 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:

$$U_{\text{prep}} = (U_A \otimes U_B)\text{CNOT}_{AB}(U'_A \otimes \mathbb{1}_B)\,. \tag{81}$$

Note that U_{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:

$$\begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i\lambda+\phi}\cos(\theta/2) \end{pmatrix}, \tag{82}$$

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. 42, 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 state of A being ρ. Likewise we have the state of A' to be ρ. 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.,

$$\langle Z\rangle_C = p_0 - p_1 = \text{Tr}(\rho^2) = P\,, \tag{83}$$

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. (79) and (80).

14.3 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. 43. 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:

$$\text{controlled-SWAP}_{CAB} = (\mathbb{1}_C \otimes \text{CNOT}_{BA})\text{Toffoli}_{CAB}(\mathbb{1}_C \otimes \text{CNOT}_{BA}) \tag{84}$$

and then decomposing the Toffoli gate, as in Ref. [91].

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]).

Our results are as follows. For the example given in Eq. (76), IBM's 5-qubit simulator with 40960 trials gave:

$$e_1 = 1.57492\,, \quad e_2 = 0.102965 \quad \text{(IBM's simulator)}\,. \tag{85}$$

A comparison with Eq. (77) shows that IBM's simulator essentially gave the correct answer. On the other hand, IBM's 5-qubit quantum computer with 40960 trials gave:

$$e_1 = 0.838943 + 0.45396i\,, \quad e_2 = 0.838943 - 0.45396i \quad \text{(IBM's Quantum Computer)}\,. \tag{86}$$

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 for this problem.

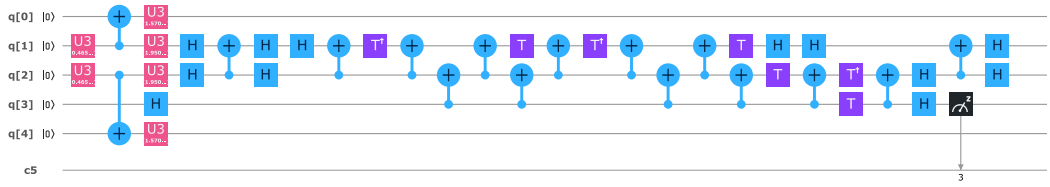


Fig. 43. Actual circuit for quantum PCA implemented on IBM's 5-qubit simulator and quantum computer. The first three time slots in the score correspond to the state preparation step of the algorithm, and the subsequent time slots correspond to the purity calculation step. Due to connectivity reasons, 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. (84), and then decomposing the Toffoli gate into CNOT gates [91].

15 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 [102]. It approximates the hyperplane finding procedure of SVM by solving the following linear equation:

$$\begin{bmatrix} 0 & \vec{1}^T \\ \vec{1} & \mathbf{K} + \gamma^{-1}\mathbf{1} \end{bmatrix} \begin{bmatrix} b \\ \vec{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \vec{y} \end{bmatrix} . \tag{87}$$

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 $\vec{w}$ where $\vec{w} = \sum_{j=1}^{M} \alpha_j \vec{x}_j$. Various definitions for the kernel matrix are available, but the quantum SVM [84] uses linear kernel: $K_{ij} = \vec{x}_i \cdot \vec{x}_j$. Classically, the complexity of the LS-SVM is $O\big(M^2(M+N)\big)$.

The quantum version of SVM performs the LS-SVM algorithm using quantum computers [84]. It calculates the kernel matrix using the quantum algorithm for inner product [69] on quantum random access memory [50], solves the linear equation using a quantum algorithm for solving linear equations [50], and performs the classification of a query data using the trained qubits with a quantum algorithm [84]. The overall complexity of the quantum SVM is $O\big(\log NM\big)$.

The algorithm is summarized below:

Algorithm 13 Quantum SVM [84]

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 [69].

Step 2. Solve linear equation Eq. (87) and find $|b, \vec{\alpha}\rangle$ using a quantum algorithm for solving linear equations [50] (training step).

Step 3. Perform classification of the query data $\vec{x}$ against the training results $|b, \vec{\alpha}\rangle$ using a quantum algorithm [84].

The inner product calculation to compute the kernel matrix cannot be done reliably in the currently available quantum processors. The other important part of the algorithm, which is linear system solving, can be quantized and has been dealt with in Section IV.

16 QUANTUM SIMULATION OF THE SCHRÖDINGER EQUATION

16.1 Problem definition and background

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

$$i\hbar \frac{\partial}{\partial t}\psi(x,t) = \hat{H}\psi(x,t) = \left[\frac{\hbar^2 \hat{k}^2}{2m} + V(\hat{x})\right]\psi(x,t), \tag{88}$$

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, $\left|\psi(x, t_f)\right|^2$ is the probability of finding a quantum particle at a position x at time t_f.

Solving the Schrödinger's equation numerically is a common approach since analytical solutions are only known for a handful of systems. On a classical computer, the numerical algorithm starts by defining a wave function on a discrete grid $\psi(x_i, 0)$ with a large number of points $i \in [1, N]$. The form of the Hamiltonian, Eq. (88), allows one to split the system's evolution on a single time step Δt in two steps, which are easy to perform:

$$\psi(x_i, t_{n+1}) = e^{-iV(x_i)\Delta t}\, QFT^{\dagger}\, e^{-ik^2\Delta t}\, QFT\, \psi(x_i, t_n), \tag{89}$$

where we have assumed that $\hbar = 1$ and $m = \frac{1}{2}$. And QFT and $QFT^{\dagger}$ are the quantum Fourier transform and its inverse. 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:

$$\psi(x_i, t_f) = QFT^{\dagger}\; e^{-ik^2 t_f}\; QFT\, \psi(x_i, 0), \tag{90}$$

where $V(x) = 0$ for a free particle.

We now discuss the quantum simulation of the Schrödinger's equation similar to the one discussed in [13], [98] 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. $\left|\psi(x, t_f)\right|^2$ will be the only information we will be interested in finding out.

16.2 Algorithm description

A quantum algorithm that performs a quantum simulation of one dimensional quantum systems was presented in [13]. The procedure is outlined in Algorithm 14.

Algorithm 14 Quantum simulation of Schrödinger equation [98], [13]

Input:

- Initial wave function
- Time step size, Δt, and the number of time steps, T.
- The ability to apply phase shifts in the computational basis.
- The potential function V.

Output:

- Final wave function at time $t_f = T\delta t$ when evolved using the Schrödinger equation with the potential V.

Procedure:

Step 1. Encode the wave function on a N-point grid in a quantum state of $n = \log_2(N)$ qubits. The value of this discretized wavefunction on a grid point is equal to the value of the original wave function at the same point. The constant of proportionality must then be calculated by renormalizing the discretized wavefunction.

for $1 \leq j \leq T$ **do**

Step 2a. Apply the Quantum Fourier Transform (QFT) to go to the momentum representation.

Step 2b. Apply a diagonal phase shift of the form $|x\rangle \rightarrow e^{-ix^2\Delta t}\,|x\rangle$ in the computational basis.

Step 2c. Apply the inverse Quantum Fourier Transform to come back to the position representation.

Step 2d. Apply a phase shift of the form $|x\rangle \rightarrow e^{-iV(x)\Delta t}\,|x\rangle$.

end for

Step 3. Measure the state in the computational basis.

Figure 44 shows the following stages of the algorithm. The implementation of QFT was discussed in Section IV. Implementing phase shifts corresponding to arbitrary functions can be done using a series of controlled Z gates or CNOT gates [13]. Repeating the final measurement step over many independent runs will let us estimate the probabilities $|\psi(x, t_f)|^2$. 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 (a rectangular wave), 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 proportional to $|0, 1\rangle + |1, 0\rangle$, which can be prepared by constructing the Bell state (see Fig. 1) and applying the X gate to the first qubit.

We define the 2-qubit QFT as $QFT = \mathrm{SWAP}_{12}\, H_2\, C_2\left[\mathcal{P}_1\left(\frac{\pi}{2}\right)\right] H_1$, where $C_2\mathcal{P}$ is a phase operator controlled from the second qubit. This transformation applies phase shifts to the probability amplitudes of the qubit states 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 X_1 gate.

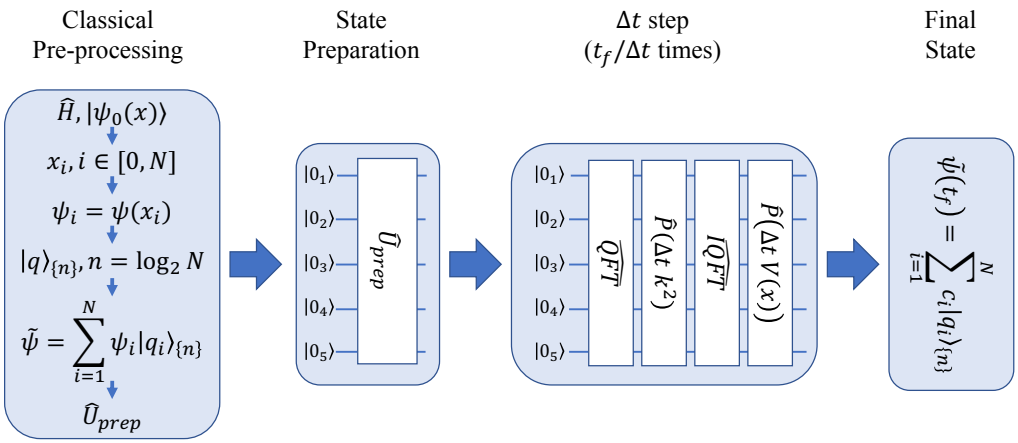


Fig. 44. The quantum simulation of the Schrödinger's equation. The first stage is a classical pre-processing that encodes the wave function to available qubits and derives a state preparation operator that takes an all-zero state of a quantum computer to a desired state. The second stage prepares an 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.

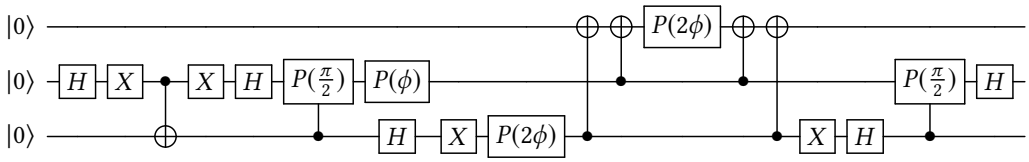


Fig. 45. The quantum circuit implementation of a 2-qubit algorithm that solves the Schrödinger's equation on a quantum computer. The initial state preparation is followed by the Quantum Fourier Transform and centering of the momentum representation. The single qubit phase shift transformations are followed 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.

The momentum encoding adopted in this discussion is $k = -\frac{1}{2}\sqrt{\frac{\phi}{\Delta t}}\left(1 + \sum_{k=1}^{n} 2^{n-k} Z_k\right)$, 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 can be individually implemented. The one qubit phase shift gate has a straightforward implementation but the two qubit phase shift gate requires an ancillary qubit according to Ref. [77], which results in a three qubit implementation on a quantum computer. This implementation is captured in Fig 45 where removing the centering of the momentum representation and the inverse QFT have been added in order to return to the coordinate representation.

16.3 Algorithm implemented on IBM's 5-qubit computer

The implementation in Fig. 46 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.

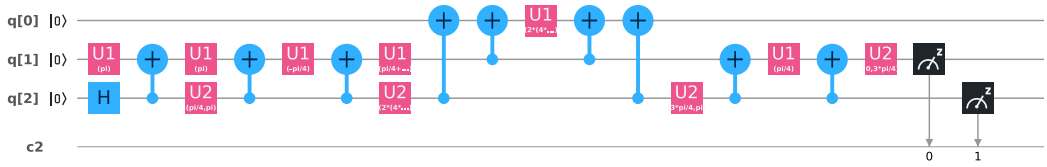


Fig. 46. The quantum circuit implementation of a 2-qubit algorithm that solves the Schrödinger's equation on the ibmqx4 quantum computer.

The circuit in Fig. 46 was run 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 was 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 returns the wave function back to the Π-function.

We 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.

17 GROUND STATE OF THE TRANSVERSE ISING MODEL

In this section the ground state of the transverse Ising model is calculated using the variational quantum eigenvalue solver, and the result is compared to the exact results. This is a hybrid method that uses alternating rounds of classical and quantum computing.

In the previous section we saw how to simulate the evolution of a single quantum particle. But often, real world phenomena are dependent on the interactions between many different quantum systems. The study of many-body Hamiltonians that model physical systems is the central theme of condensed matter physics (CMP).

Many-body Hamiltonians are inherently hard to study on classical computers as the dimension of the Hilbert space grows exponentially with the number of particles in the system. But using a quantum computer we can study these many-body systems with less overhead as the number of qubits required only grows polynomialy.

17.1 Variational quantum eigenvalue solver

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

$$\mathcal{H}|\Psi\rangle = E_g|\Psi\rangle. \tag{91}$$

Studying the ground state gives us information about the low temperature properties of the system. Once we know $|\Psi\rangle$, we can deduce the physical properties from the wave function. In this section, we will describe how to use IBM Q to find the ground state energy of the transverse Ising model. We will not be using the ibmqx4 in this section. This is because the algorithm we use will require many rounds of optimization. Each round requires us to run a circuit on the quantum computer followed by a classical optimization step on a local machine. This process can

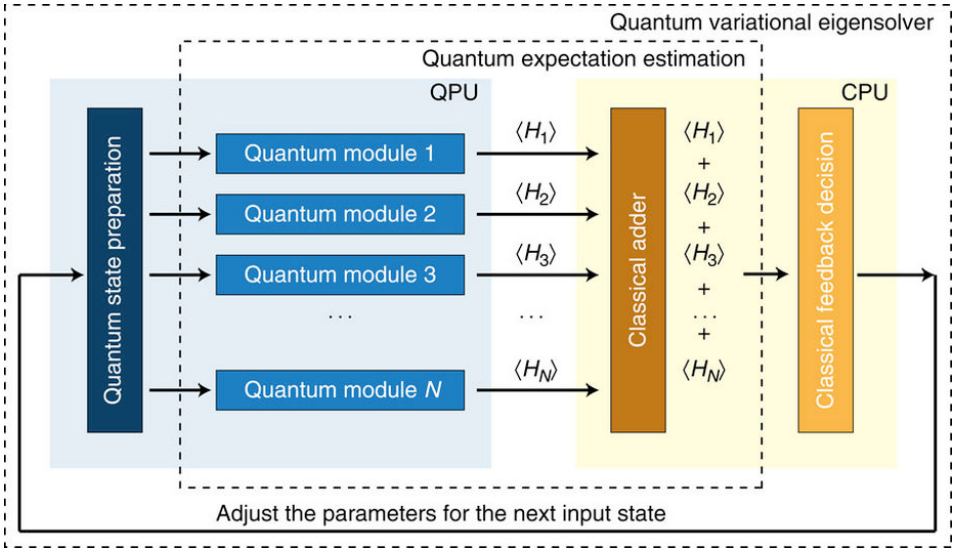


Fig. 47. Schematic view of the implementation of the variational quantum eigenvalue solver using a hybrid classical and quantum circuit. The figure is adopted from Ref. [79].

be automated easily using Qiskit. But the long queuing times in IBM Q makes repeated runs on the quantum processor impractical.

To find the eigenvalue of a Hamiltonian, we could use the quantum phase estimation algorithm that was discussed in Section IV. To do this we need the ability to perform controlled operations with the unitary $U = \exp(-i\mathcal{H}\delta t/\hbar)$, where δt is the time step. Then, by preparing different initial states $|\psi_i\rangle$ and repeating the phase estimation many times one can obtain, in principle, 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 circuits have large depth. This requires qubits with long coherence times, which are 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) [74, 79]. The basic idea is to take the advantages of both quantum and classical computers, as shown in Fig. 47. It allocates the classically easy tasks 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 of the Hamiltonian using a quantum computer, $E = \langle\psi|\mathcal{H}|\psi\rangle/\langle\psi|\psi\rangle$.
(3) Use 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 a 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 relatively simple, thus no long coherence times are not required. On the

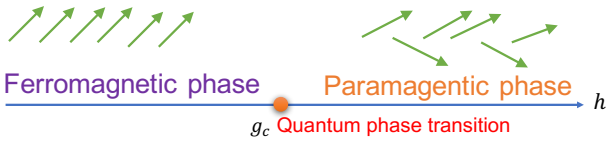


Fig. 48. Schematic view of the quantum phases described by the transverse Ising model. The arrows represent the spin configuration in the ordered and disordered phases.

other hand, VQES also has limitations. Because of its variational nature, the trial wave function needs to be prepared carefully. This usually requires physical insights into the problem. The ground state eigenvalue and eigenwave function are biased by the choice of the trial wave functions. In addition, VQES requires communications between classical and quantum computers, which could be a bottleneck for the performance.

17.2 Simulation and results

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

$$\mathcal{H} = -\sum_i \sigma_i^z \sigma_{i+1}^z - h \sum_i \sigma_i^x, \tag{92}$$

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

$$R_x \sigma^x R_x^\dagger = \sigma^x, \quad R_x \sigma^z R_x^\dagger = -\sigma^z. \tag{93}$$

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 $\langle \sigma^z \rangle \neq 0$. As h is increased, there is a quantum phase transition from the ordered phase to the disordered phase where $\langle \sigma^z \rangle = 0$, as the rotational symmetry is restored. The phase diagram is shown schematically in Fig. 48.

Using the phase diagram as a guide, first we propose a product state as a trial wave function. The wave function can be written as

$$|\psi_i(\theta_i)\rangle = \prod_i U(\theta_i)|0_i\rangle. \tag{94}$$

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

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

where θ_i are the variational parameters. Here we have used the Bloch sphere representation for a qubit state. For the TIM, we calculate the expectation value of

$$E_{J,i} = -\langle \psi | \sigma_i^z \sigma_{i+1}^z | \psi \rangle, \quad E_{Z,i} = -\langle \psi | \sigma_i^x | \psi \rangle. \tag{95}$$

The quantum circuit to perform the preparation of the state and calculation of the expectations value are shown in Fig. 49(a) andFig. 49(b) . We have

$$E_{J,i} = -[P(q_i = 0) - P(q_i = 1)][P(q_{i+1} = 0) - P(q_{i+1} = 1)], \tag{96}$$

$$E_{Z,i} = -[P(q_i = 0) - P(q_i = 1)], \tag{97}$$

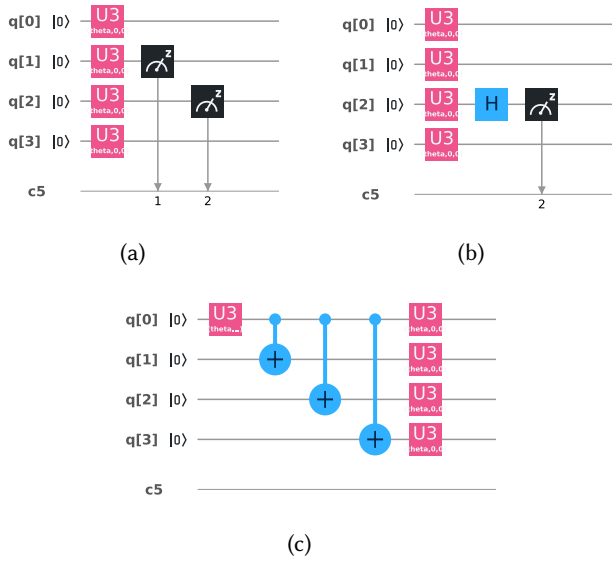


Fig. 49. Quantum circuits to prepare the trial wave-functions. The single qubit unitaries in the text can be implemented using available gates in IBM Q. The first two circuits prepare unentangled trial states. Circuit (a) can be used to measure $\langle\psi|\,\sigma_2^z\sigma_3^z\,|\psi\rangle$. Circuit (b) can be used to measure the $\langle\psi|\,\sigma_3^x\,|\psi\rangle$. Circuit (c) prepares the entangled trial state.

where $P(q_i = 0, 1)$ is the measured probability for the qubit q_i in the $|0\rangle$ or $|1\rangle$ state. As we mentioned before, the communication bottleneck prevented us from implementing this on `ibmqx4`. We ran the code using the quantum simulator in Qiskit. The comparison of the results obtained from quantum simulation and analytical results are shown in Fig. 50. 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 the 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

$$|\psi_i(\theta_i)\rangle = \alpha|\Psi_i(\theta_i)\rangle + \beta R_x(\pi)|\Psi_i(\theta_i)\rangle. \tag{98}$$

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 [80]. The state in Eq. (98) can be prepared easily using the circuit in Fig. 49(c). 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

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

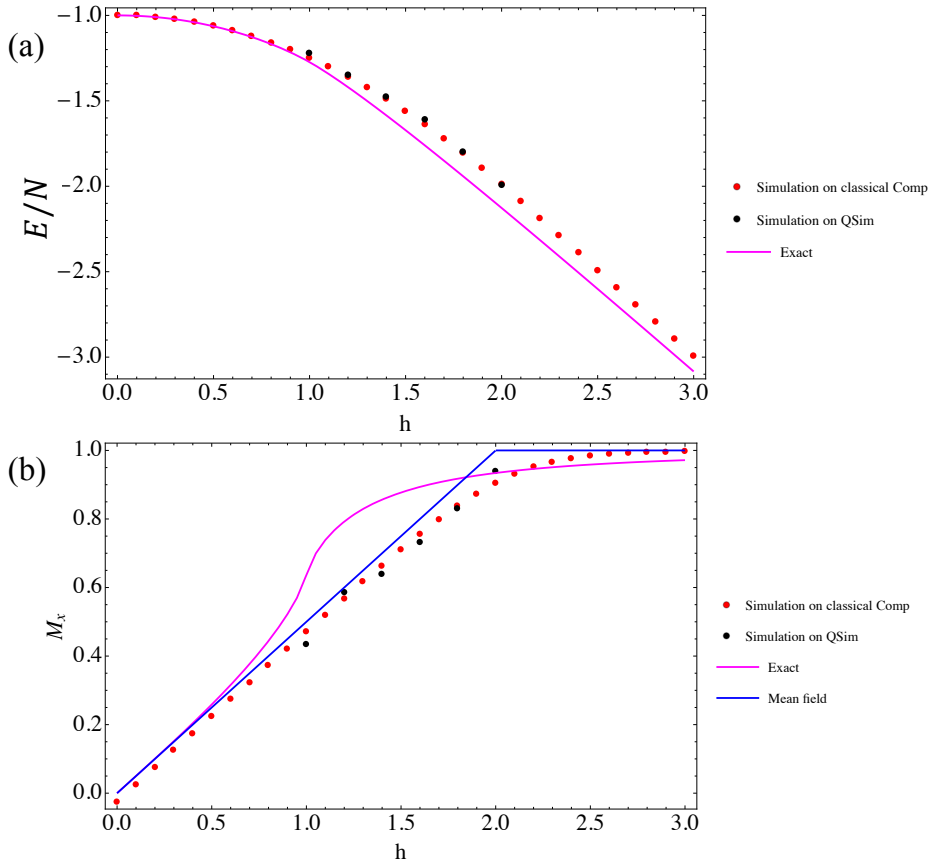


Fig. 50. Comparison of the ground state energy (a) and average magnetization (b) $M_x = \langle\psi|\sum_i \sigma_i^x|\psi\rangle/N$ obtained by using the trial wave functions in Eq. (94) and the exact results. Here we have used the periodic boundary condition. The simulations are run both on the quantum simulator (black symbols) and classical computers (red symbols). The mean-field results (blue line) are also displayed for comparison.

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. (98). 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. 51, the new trial function nearly reproduces the exact results in the whole magnetic field region and improves upon the product state trial function.

18 QUANTUM PARTITION FUNCTION

18.1 Background on the partition function

Calculation or approximation of the partition function is a sub-step of inference problems in Markov networks [63]. Even for small networks, this calculation becomes intractable. Therefore an efficient

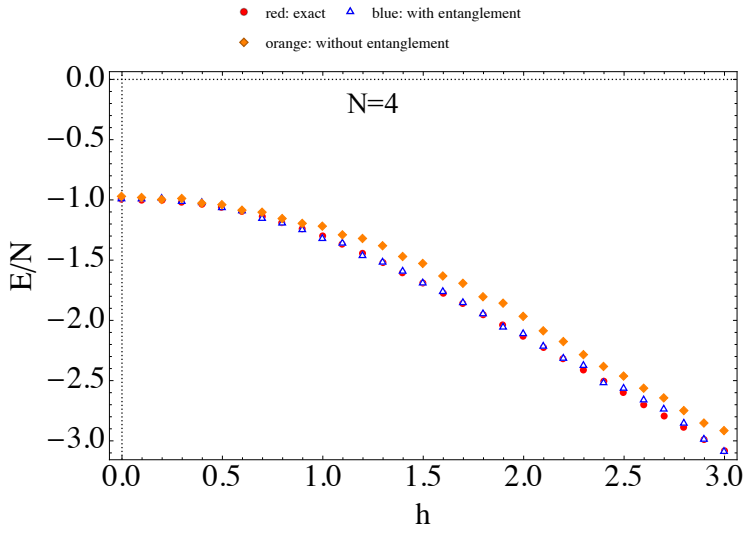


Fig. 51. (color online) Comparison of the ground state energy obtained by using the trial wave functions in Eqs. (94) and (98) and the exact result. Here we have used the periodic boundary condition. The number of spins is 4.

quantum algorithm for the partition function would make many problems in graphical model inference and learning tractable and scalable; the same holds for other problems in computational physics [7, 46, 48, 49].

The partition function is of particular interest for calculating probabilities from graphical models such as Markov random fields [63]. 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 classical Ising model. In the classical 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

$$H(\sigma) = -\sum_{i \sim j} J_{ij} \delta_{\sigma_i, \sigma_j}, \tag{99}$$

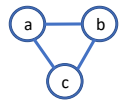
where $i \sim j$ indicates that there exists an edge between vertices i and j; and where $\delta_{\sigma_i, \sigma_j} = 1$ if $\sigma_i = \sigma_j$ and 0 otherwise.

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

$$P(\sigma) = \frac{1}{Z} e^{-\beta H(\sigma)}, \tag{100}$$

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*:

$$Z = \sum_{\{\sigma\}} e^{-\beta H(\sigma)}, \tag{101}$$



(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. 52. (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)

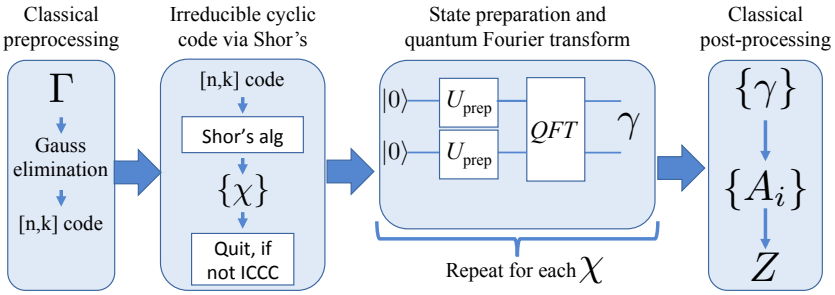


Fig. 53. Overview of the quantum partition function algorithm.

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 a large 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 exists one [49].

18.2 A 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 52a, 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 Fig. 52b.

We plug the value of the Hamiltonian for each of the q^n configurations into the partition function given in Eq. (101) to get the normalization constant:

$$Z = 2e^{\beta(J_{ab}+J_{bc}+J_{ac})} + 2e^{\beta J_{ab}} + 2e^{\beta J_{bc}} + 2e^{\beta J_{ac}}. \tag{102}$$

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

$$Z = 2e^{3\beta} + 6e^{\beta}. \tag{103}$$

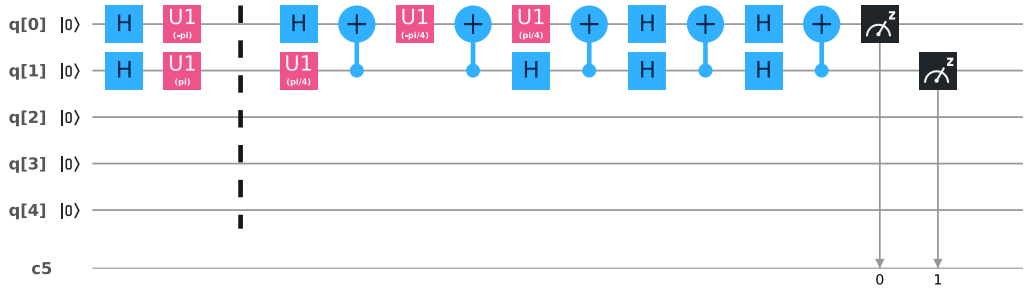


Fig. 54. Circuit for preparing the first two qubits and quantum Fourier transform on 2 qubits.

18.3 Calculating the quantum partition function

An efficient quantum algorithm for the partition function is given by [49] 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.

18.4 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 this requirement. In the fully-connected 3-vertex Ising model example given, the input to QFT2 is $q[0] = |+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $q[1] = |-\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$. In the sample score shown in Fig. 54, these qubits are prepared before the barrier. The QFT2 algorithm, as given by the Qiskit Tutorial provided by IBM[3], 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.

The result from simulating 1000 shots gives $P(\gamma = 1) = 0.47$ and $P(\gamma = 3) = 0.53$. The results from running on the actual hardware are, $P(\gamma = 0) = 0.077$, $P(\gamma = 1) = 0.462$, $P(\gamma = 2) = 0.075$, and $P(\gamma = 3) = 0.386$. We can threshold the low-probability values of gamma, ensuring that no more

than the maximum number (as given in [49]) of distinct values of gamma remain. These gammas are then plugged into the calculation of the weight spectrum and the partition function.

19 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.

19.1 Single qubit state preparation

As discussed before, a single qubit quantum state $|\psi\rangle$ is represented as a superposition of $|0\rangle$ and $|1\rangle$ states $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, where $|\alpha|^2 + |\beta|^2 = 1$. The sizes $|\alpha|^2$ and $|\beta|^2$ represent the probability of $|\psi\rangle$ being $|0\rangle$ or $|1\rangle$. 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.

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

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

This gate is directly available in IBM Q and is implemented in a composite fashion on `ibmqx4` at the hardware level. 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 $R_z(\beta) = \mathrm{diag}(e^{i\beta/2}, e^{-i\beta/2})$ 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.

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 a distance of 0.024 of a desired gate. Thus, to within an accuracy of about

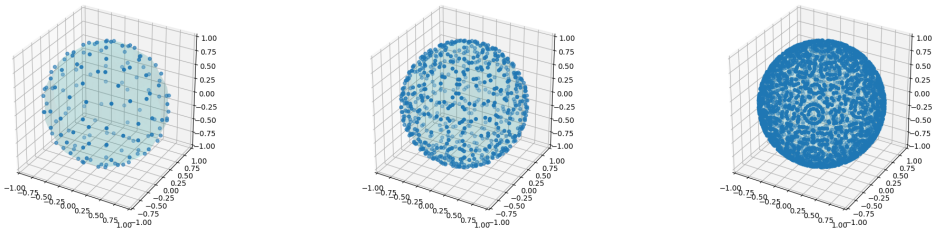


Fig. 55. Possible exact state initializations using 10, 15, and 20 gates. With 20 gates, every point on the sphere is within a distance of approximately 0.072 of an exactly obtainable state. With 30 gates, every point is within 0.024

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.

19.2 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

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

In a Schmidt decomposition, we obtain such a state by strategically choosing two orthonormal bases $\left|\xi_j\right\rangle, \left|\varphi_j\right\rangle$ 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 $\left|\xi_j\right\rangle$ and $\left|\varphi_j\right\rangle$ 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 computational 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

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

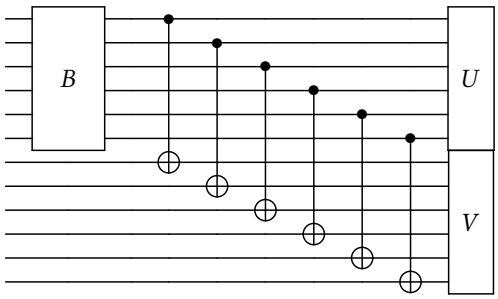


Fig. 56. Schmidt decomposition.

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

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

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 $|\psi\rangle = \sum_{i,j=1}^{2^n} a_{ij}|e_i\rangle\left|e_j\right\rangle$ 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

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

Multiplying on the left by $\langle e_i|\left\langle e_j\right|$ 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 $v_{jk} = \left\langle e_j\right|V|e_k\rangle$ 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

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

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

$$V = AU^{T\dagger}\text{diag}(\lambda_1, \ldots, \lambda_n)^{-1}.$$

The matrix U is unitary, and one easily checks that V is therefore also unitary. Moreover $\sum_i |b_{i1}|^2 = \text{Tr}(A^\dagger A) = \sum_i |a_{ij}|^2 = 1$, and so the b_{i1}'s are representative of an n-qubit state and can be taken as the first column of B. Readers familiar with singular value decompositions (SVD) will recognize that the Schmidt decomposition of a bipartite state is essentially the SVD of the coefficient matrix A associated with the state. The λ_i coefficients being the singular values of A.

19.3 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 magnitudes 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 in Fig. 57.

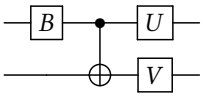


Fig. 57. Circuit for two qubit-state preparation. The choice of U, V, and B are covered comprehensively in the Schmidt decomposition description.

19.4 Two-qubit gate 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 given in Fig. 58.

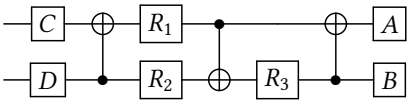


Fig. 58. Circuit implementation of an arbitrary two qubit gate.

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

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

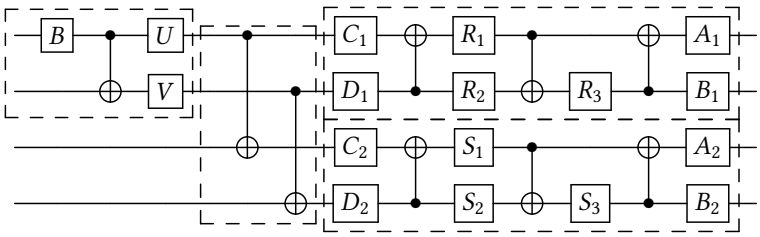


Fig. 59. Circuit for four qubit-state preparation. The four phases of the circuit are indicated in dashed boxes.

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 $\widetilde{U} = E^\dagger UE$ and $\widetilde{V} = E^\dagger VE$. 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

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

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

19.5 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 shown in Fig. 59. 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.

The results of this circuit implemented on a quantum processor are given in Fig. 60. While the results when implemented on a simulator are given in Fig. 61.

20 QUANTUM TOMOGRAPHY

20.1 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, 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 state 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.*

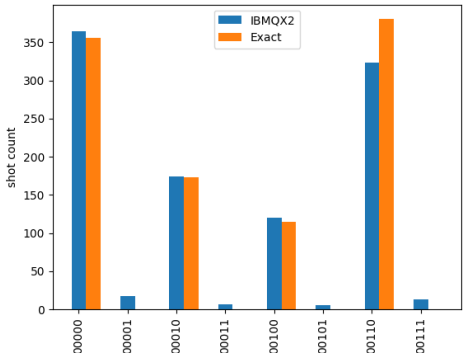


Fig. 60. Verification of 4 qubit state preparation on `ibmqx2` which is a 5 qubit machine. The last qubit is not used in the circuit. The above histogram shows that, the state prepared in `ibmqx2` has nonzero overlaps with basis states that are orthogonal to the target state to be prepared.

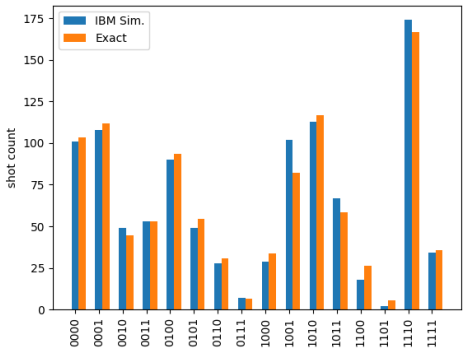


Fig. 61. Verification of the quantum circuit for four qubit-state preparation. The differences in the exact and the simulator results are due to statistical fluctuations arising from the probabilistic nature of quantum measurement. They will become closer to each other when the number of samples are increased.

Moreover 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 to use the *density matrix* representation of quantum states. We briefly discussed this representation in the context of quantum principal component analysis in Section XIV.

Let us denote p_i the probability of occurrence of the state $|\psi_i\rangle$. The density matrix of this ensemble is given by,

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|. \tag{104}$$

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 $\rho \geqslant 0$, i.e., all eigenvalues are either positive or zero.

In a popular setting for quantum tomography [77], 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 occurrences 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.

20.2 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 $\widehat{\rho} \geqslant 0$ (i.e., might contain negative eigenvalues) [56].

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

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

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 [54, 56]:

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

This is a convex problem that outputs a positive semidefinite (PSD) solution $\widehat{\rho} \geqslant 0$. 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. [90]. 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 [18], or corrections to the basic ML objective, see e.g., the so-called Hedged ML [17]. 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.

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 [51] in the

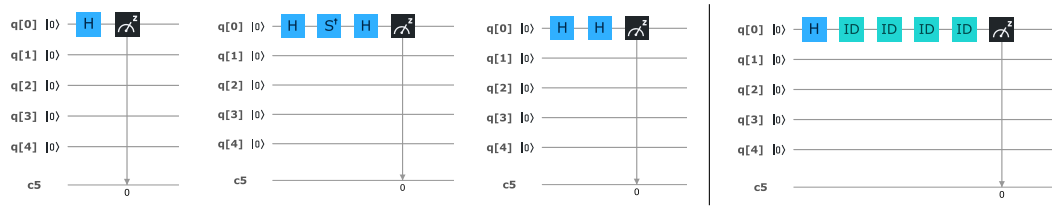


Fig. 62. Left: measurements of the single qubit state after the application of the Hadamard gate, in z, y and x basis. Right: experimental setup for testing the effects of decoherence.

case of low-rank solutions, and matrix product states [33] or neural networks based approaches [104] for pure states with limited entanglement, *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.

20.3 Implementation of the 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 [97] to the space of PSD matrices. In what follows, we apply quantum tomography to study the performance of the IBM Q.

20.3.1 Warm-up: Hadamard gate. Let us start with a simple one-qubit case of the Hadamard gate, see Fig. 62, Left. 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 $|+\rangle_x$.

Instead, we obtain 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):

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

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 (Fig. 62, Right) which forces an additional waiting on the system, and hence promotes decoherence

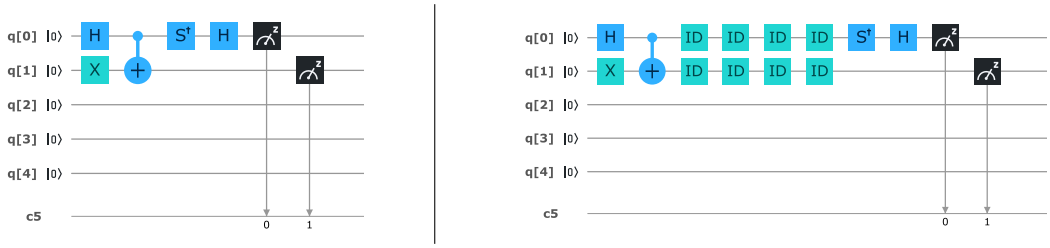


Fig. 63. Left: example of a measurement of the two-qubit maximally entangled state created with the combination of H, X and $CNOT$ gates in the yz basis. Right: experimental setup for testing the effects of decoherence.

of the state. When applying 18 identity matrices, we obtain the following decomposition for $\widehat{\rho}$

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

while application of 36 identity matrices results in

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

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

20.3.2 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 Fig. 63, 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:

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

Our test of decoherence implemented using 18 identity matrices (see Figure 63, Right) shows that the probability of the "original" entangled state decreases to 0.79:

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

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.

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, which is the subject of the next section.

21 TESTS OF QUANTUM ERROR CORRECTION IN IBM Q

In this section, we study whether quantum error correction (QEC) can improve computation accuracy in `ibmqx4`. The practical answer to this question seems to be "No". Although some error correction effects are observed in `ibmqx4`, improvements are not exponential and get completely spoiled by errors induced by extra gates and qubits needed for the error correction protocols.

21.1 Problem definition and background

As we have seen throughout this review, the quality of computation on actual quantum processors is degraded by errors in the system. This is because currently available chips are not *fault tolerant*. It is widely believed that once the inherent error rates of a quantum processor is sufficiently lowered, fault tolerant quantum computation will be possible using quantum error correction (QEC). The current error rates of the IBM Q machines are not small enough to allow fault tolerant computation. We refer the reader to a survey and introduction on QEC [37], while at the same time offering an alternative point of view that we support with a few experiments on the IBM chip. The central idea of QEC is to use entanglement to encode quantum superposition in a manner which is robust to errors. The exact encoding depends upon the kind of errors we want to protect against. In this section we will look at a simple encoding that will protect against bit flip errors. Here we encode a single qubit state,

$$|\psi\rangle = C_0|0\rangle + C_1|1\rangle, \tag{105}$$

using an entangled state, such as

$$|\psi\rangle = C_0|0\rangle^{\otimes n_q} + C_1|1\rangle^{\otimes n_q}, \tag{106}$$

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 (106). Measurements can then be used to fix a single qubit error using, for instance, a majority voting strategy. More complex errors are assumed to be exponentially suppressed, which can be justified if qubits experience independent decoherence sources.

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 (106), 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. Hence, gate-induced errors may not be treatable by standard error correction strategies when transitions are made between arbitrary unknown quantum state.

To explore 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

$$|u\rangle = \cos(\delta\phi)|0\rangle + \sin(\delta\phi)|1\rangle, \tag{107}$$

Fig. 64. Quantum circuit that creates the state $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ then applies 16 T-gates that are equivalent to the identity operation, and then applies the gate that transforms the entangled state into the trivial state $|0\rangle$.

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

$$P \approx (\delta\phi)^2. \tag{108}$$

The value $1 - P$ is called the fidelity of the gate. In IBM chip it is declared to be 0.99 at the time of writing, 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.

In order to reduce this error, we can attempt to work with the 3-qubit version of the states in Eq. (106). For example, let us consider the desired gate that transfers states

$$|\pm\rangle = (|000\rangle \pm |111\rangle)/\sqrt{2}, \tag{109}$$

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 (109). The error in this rotation angle leads to the final state

$$|u\rangle = \cos(\delta\phi)|000\rangle + \sin(\delta\phi)|111\rangle, \tag{110}$$

i.e., this particular error cannot be treated with majority voting using our scheme 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 the qubits are encoded. With nine qubits, we could protect the sign in Eq. 110 but this was beyond our hardware capabilities.

Based on these thoughts, traditional QEC may not 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 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.

21.2 Test 1: errors in single qubit control

First, let us perform trivial operation shown in Fig. 64: we create a superposition of two qubit states

$$|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}, \tag{111}$$

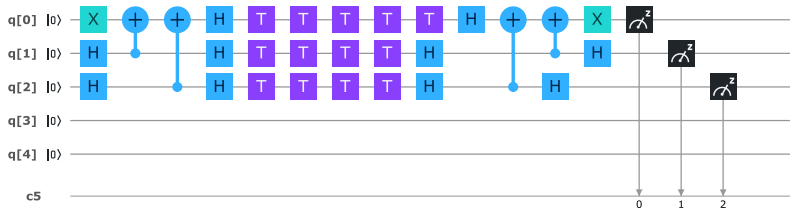


Fig. 65. Quantum circuit that creates state $|-\rangle = (|000\rangle - |111\rangle)/\sqrt{2}$ then applies 16 T-gates that are equivalent to the identity operation, and then applies the gate that transforms the entangled GHZ state back into the trivial state $|000\rangle$. Measurements that return 1 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 $|111\rangle$.

then apply many gates that altogether do nothing, i.e., they just bring the qubit back to the superposition state (111). We need those gates just to accumulate some error while the 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.

21.3 Test 2: errors in entangled 3 qubits control

Next, we consider the circuit in Fig. 65 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 the 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 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, the 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 $1 - P_{000} = 0.16$.

21.4 Discussion

Comparing results of the tests without and with QEC, we find that the implementation of a simple version of 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 the 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. 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 that probabilities of single bit flip errors were 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, an 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 a 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 conventional 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 [95]. So, they must be developed even if problems with quantum error correction prove detrimental for conventional quantum computing schemes at achievable hardware quality.

ACKNOWLEDGMENTS

We would like to acknowledge the help from numerous readers who pointed out errors and misprints in the earlier version of the manuscript. The code and implementations accompanying the paper can be found at https://github.com/lanl/quantum_algorithms.

REFERENCES

[1] ibmq-device-information. https://github.com/Qiskit/ibmq-device-information/tree/master/backends/tenerife/V1. Accessed: 14-12-2019.

[2] 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.

[3] Héctor Abraham, Ismail Yunus Akhalwaya, Gadi Aleksandrowicz ...., and yotamvakninibm. Qiskit: An open-source framework for quantum computing, 2019.

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

[5] Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007.

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

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

[8] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[9] Dave Bacon, Isaac L Chuang, and Aram W Harrow. The quantum schur and clebsch-gordan transforms: I. efficient qudit circuits. pages 1235–1244, 2007.

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

[11] 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.

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

[13] Giuliano Benenti and Giuliano Strini. Quantum simulation of the single-particle Schrödinger equation. *American Journal of Physics*, 76(7):657–662, 2008.

[14] 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.

[15] 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.

[16] Dominic W Berry, Graeme Ahokas, Richard Cleve, and Barry C Sanders. Efficient quantum algorithms for simulating sparse hamiltonians. *Communications in Mathematical Physics*, 270(2):359–371, 2007.

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

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

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

[20] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics*, 46(4-5):493–505, 1998.

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

[22] Carlos Bravo-Prieto, Ryan LaRose, Marco Cerezo, Yigit Subasi, Lukasz Cincio, and Patrick J Coles. Variational quantum linear solver: A hybrid algorithm for linear systems. *arXiv preprint arXiv:1909.05820*, 2019.

[23] H. Buhrman and R. Spalek. Quantum verification of matrix products. *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 880–889, 2006.

[24] 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.

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

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

[27] Lukasz Cincio, Yiğit Subaşı, Andrew T Sornborger, and Patrick J Coles. Learning the quantum algorithm for state overlap. *New Journal of Physics*, 20(11):113022, 2018.

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

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

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

[31] 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.

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

[33] 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.

[34] Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Vazirani. *Algorithms*. McGraw-Hill, Inc., New York, NY, USA, 2008.

[35] M. Dehn. Über unendliche diskontinuierliche gruppen. *Mathematische Annalen*, 71(1):116–144, Mar 1911.

[36] 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.

[37] Simon J Devitt, William J Munro, and Kae Nemoto. Quantum error correction for beginners. *Reports on Progress in Physics*, 76(7):076001, 2013.

[38] B. L. Douglas and J. B. Wang. Efficient quantum circuit implementation of quantum walks. *Physical Review A*, 79(5):052335, 2009.
[39] 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.
[40] Christoph Durr and Peter Hoyer. A quantum algorithm for finding the minimum. *arXiv preprint quant-ph/9607014*, 1996.
[41] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19 (2):248–264, 1972.
[42] Nayak F. Magniez A, J. Roland, and M. Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011.
[43] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014.
[44] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.
[45] R. Freivalds. Fast probabilistic algorithms. *In Proc. of 8th Symp. on Math. Foundations of Computer Science*, pages 57–69, 1979.
[46] Silvano Garnerone, Annalisa Marzuoli, and Mario Rasetti. Efficient quantum processing of 3-manifold topological invariants. *arXiv preprint quant-ph/0703037*, 2007.
[47] Iulia M Georgescu, Sahel Ashhab, and Franco Nori. Quantum simulation. *Reviews of Modern Physics*, 86(1):153, 2014.
[48] Joseph Geraci. A new connection between quantum circuits, graphs and the ising partition function. *Quantum Information Processing*, 7(5):227–242, 2008.
[49] 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.
[50] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. 100:160501, 04, 2008.
[51] 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.
[52] 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.
[53] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009.
[54] Zdenek Hradil. Quantum-state estimation. *Physical Review A*, 55(3):R1561, 1997.
[55] IBM Corporation. IBM Quantum Experience. Published online at https://quantumexperience.ng.bluemix.net, 2016. Accessed: 12/01/2017.
[56] Daniel F. V. James, Paul G. Kwiat, William J. Munro, and Andrew G. White. Measurement of qubits. *Phys. Rev. A*, 64:052312, 2001.
[57] Sonika Johri, Damian S Steiger, and Matthias Troyer. Entanglement spectroscopy on a quantum computer. *Physical Review B*, 96(19):195136, 2017.
[58] Stephan Jordan. Quantum Algorithm Zoo. Published online at https://math.nist.gov/quantum/zoo/, 2011. Accessed: 3/18/2018.
[59] Stephen P. Jordan. Fast quantum algorithms for approximating some irreducible representations of groups . pages 1–21, 2009.
[60] Petteri Kaski. Eigenvectors and spectra of cayley graphs, 2002.
[61] J. Kempe. Quantum random walks - an introductory overview. *Contemporary Physics*, 44(4):307–327, 2003.
[62] V. Kendon. Where to quantum walk. pages 1–13, 2011.
[63] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. Adaptive Computation and Machine Learning. MIT Press, 2009.
[64] 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.
[65] Thaddeus D Ladd, Fedor Jelezko, Raymond Laflamme, Yasunobu Nakamura, Christopher Monroe, and Jeremy Lloyd O'Brien. Quantum computers. *Nature*, 464(7285):45, 2010.
[66] R. LaRose, A. Tikku, É. O'Neel-Judy, L. Cincio, and P. J. Coles. Variational quantum state diagonalization. *npj Quantum Information*, 5(1):57, 2019.
[67] R. J. Lipton and K. W. Regan. Quantum algorithms via linear algebra. 2014.
[68] Seth Lloyd, Silvano Garnerone, and Paolo Zanardi. Quantum algorithms for topological and geometric analysis of data. *Nature Communications*, 2015.
[69] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum algorithms for supervised and unsupervised machine learning. *arXiv preprint arXiv:1307.0411*, 2013.
[70] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631–633, 2014.

[71] Neil B Lovett, Sally Cooper, Matthew Everitt, Matthew Trevers, and Viv Kendon. Universal quantum computation using the discrete-time quantum walk. *Physical Review A*, 81(4):042330, 2010.
[72] Frederic Magniez, Miklos Santha, and Mario Szegedy. Quantum algorithms for the triangle problem. *SIAM J. Comput.*, pages 413–424, 2007.
[73] Enrique Martin-Lopez, Anthony Laing, Thomas Lawson, Roberto Alvarez, Xiao-Qi Zhou, and Jeremy L O'brien. Experimental realization of shor's quantum factoring algorithm using qubit recycling. *Nature photonics*, 6(11):773–776, 2012.
[74] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2):023023, 2016.
[75] Ashley Montanaro. Quantum algorithms: an overview. *npj Quantum Information*, 2:15023, 2016.
[76] Michele Mosca. Quantum algorithms. In *Computational Complexity*, pages 2303–2333. Springer, 2012.
[77] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, United Kingdom, 2016. 10th Anniversary Edition.
[78] Karl Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 6*, 2(11):559–572, 1901.
[79] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5:ncomms5213, July 2014.
[80] Martin Plesch and Časlav Brukner. Quantum-state preparation with universal gate decompositions. *Phys. Rev. A*, 83:032302, 2011.
[81] Carl Pomerance. A tale of two sieves. *Notices Amer. Math. Soc*, 43:1473–1485, 1996.
[82] John Preskill. Quantum computing and the entanglement frontier. *Rapporteur talk at the 25th Solvay Conference on Physics, 19-22 October 2011*.
[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] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical review letters*, 113(13):130503, 2014.
[85] E. Rieffel and W. Polak. Quantum computing: A gentle introduction. 2011.
[86] 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.
[87] Mehdi Saeedi and Igor L Markov. Synthesis and optimization of reversible circuits - a survey. *ACM Computing Surveys (CSUR)*, 45(2):21, 2013.
[88] Miklos Santha. Quantum walk based search algorithms. In *International Conference on Theory and Applications of Models of Computation*, pages 31–46. Springer, 2008.
[89] N. Santhi. Quantum Netlist Compiler (QNC) software repository, November 2017. Applied for LANL LACC authorization for unlimited open-source release, December 2017.
[90] Jiangwei Shang, Zhengyun Zhang, and Hui Khoon Ng. Superfast maximum-likelihood reconstruction for quantum tomography. *Physical Review A*, 95(6):062336, 2017.
[91] Vivek V. Shende and Igor L. Markov. On the CNOT-cost of TOFFOLI gates. *Quant. Inf. Comp.*, 9(5-6):461–486, 2009.
[92] Neil Shenvi, Julia Kempe, and K Birgitta Whaley. Quantum random-walk search algorithm. *Physical Review A*, 67(5):052307, 2003.
[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] Nikolai A Sinitsyn. Computing with a single qubit faster than the computation quantum speed limit. *Physics Letters A*, 382(7):477–481, 2018.
[96] Robert S. Smith, Michael J. Curtis, and William J. Zeng. A practical quantum instruction set architecture, 2016.
[97] 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.
[98] Rolando D Somma. Quantum simulations of one dimensional quantum systems. *Quantum Information & Computation*, 16(13-14):1125–1168, 2016.
[99] Robert Spalek et al. *Quantum algorithms, lower bounds, and time-space tradeoffs*. ILLC,Amsterdam, 2006.
[100] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, (13):354–356, 1969.
[101] Yiğit Subaşı, Rolando D Somma, and Davide Orsucci. Quantum algorithms for systems of linear equations inspired by adiabatic quantum computing. *Physical review letters*, 122(6):060504, 2019.

[102] J. A. K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Process. Lett.*, 9(3):293–300, June 1999.

[103] IBM QX Team. IBM Q experience backend information. http://github.com/QISKit/ibmqx-backend-information, 2017. Last accessed: 12 December, 2017.

[104] Giacomo Torlai, Guglielmo Mazzola, Juan Carrasquilla, Matthias Troyer, Roger Melko, and Giuseppe Carleo. Neural-network quantum state tomography. *Nature Physics*, 14(5):447, 2018.

[105] Jaw-Shen Tsai. Toward a superconducting quantum computer. *Proceedings of the Japan Academy, Series B*, 86(4):275–292, 2010.

[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*, 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. *Quantum Information and Computation II*, 5436:285–296, 2004.

[108] Guifré Vidal. Efficient classical simulation of slightly entangled quantum computations. *Physical review letters*, 91(14):147902, 2003.

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

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

[111] 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.