Quantum Circuit Decomposition

Adrien Ecoffet

Context: I wrote this paper as a final project for the Matrix Theory course of the Johns Hopkins University master's in Applied and Computational Mathematics. I think it provides a more intuitive introduction to quantum circuit decomposition than the other works I am familiar with, so I am putting it online with approval from the professor. I have eliminated most references to the course textbook, Shores [1], but it may still be useful as a reference since I sometimes assume familiarity with the course material.

Abstract

Quantum computing uses quantum mechanical properties of matter to perform operations unavailable to classical computers. Two important representations for an n-qubit quantum algorithm are the matrix representation, which represents the quantum state as a complex unit vector and the algorithm as a unitary matrix, and the circuit representation, which represents the algorithm as a circuit of quantum gates. The matrix representation is mathematically useful, but the circuit representation is closer to what can be implemented on quantum hardware, so it is important to be able to convert between the two. While going from gates to matrices is trivially done by multiplying the matrix representations of the gates, going from a unitary to a quantum circuit is the subject of a significant field of study known as quantum circuit decomposition. Here, I present the decomposition algorithm presented by Cybenko [2]. After a brief introduction to quantum computing and the two representations, I show how the QR decomposition can be implemented using Givens operations, and how applying it to unitary matrices results in a product of simple unitary matrices. I then explain the structure of quantum gate matrices, and show that although Givens operations do not always match it, they can be decomposed into a product of unitary matrices that do. I finally reduce the set of necessary gates to a small universal gate set.

1. Introduction

Quantum computing promises to greatly speed up computations by harnessing the quantum mechanical properties of matter. Shor's quantum factorization algorithm [3] can factorize large integers in polynomial time, a fact which could be used to break many common encryption schemes [4]. Grover's quantum search algorithm can perform an exhaustive search in $O(\sqrt{N})$ operations rather than the $O(N)$ required on classical computers [5]. Further, quantum computers could efficiently simulate any quantum-mechanical system [6], so that the properties of new drugs or materials could be simulated without the compounds needing to be synthesized and tested, thus greatly accelerating progress in medicine, material science, and many other fields [4].

Practically implementing quantum computers is a subject of intense study. One of the difficulties involved is the gap between the mathematical representation of quantum algorithms and representations that are practically implementable. Mathematically, a quantum algorithm can be represented as a large unitary matrix, and running the algorithm is equivalent to multiplying that matrix with an input vector. There is often no obvious way to implement such large operations in hardware, however, so it is necessary to decompose the algorithm into a series of elementary operations called quantum gates, producing an implementable quantum circuit. Here, I present the decomposition algorithm described by Cybenko [2], with further details coming mainly from Barenco et al. [7]. Much of the background on quantum computing comes from Matuschak and Nielsen [4], which I highly recommend as an introduction.

2. Background on Quantum Computing

Similar to bits in classical computing, quantum computing uses qubits (often also called "bits" when there is no ambiguity [7]), whose measurable values can be 0 or 1. Qubits are typically implemented as particles (e.g. an atom or photon) which can be in one of two states, one of which is taken as representing 0 and the other 1. While the state of a classical computer is always a well defined sequence of bits such as `01` or `11`, the state of a quantum computer can be a *superposition* of multiple sequences of bits, so that it could be in states `01` and `11` "at the same time."

$$\begin{array}{c} \left[\begin{array}{c} 0 \\ \sqrt{0.3}+i\sqrt{0.06} \\ 0 \\ 0.8 \end{array}\right] \end{array} \begin{array}{c} \texttt{00} \\ \texttt{01} \\ \texttt{10} \\ \texttt{11} \end{array}$$

Figure 1: An amplitude vector (the values on the right are the states each amplitude corresponds to).

What does it mean for a quantum computer to be in state `01` and `11` at the same time? For one thing, it affects the outcome of *measurement*: we could be in a situation where if we tried to measure our state, we would have a 36% chance of measuring `01` and a 64% chance of measuring `11`. If we happened to measure, say, `01`, and measured again, we would keep measuring `01` from then on.

From the description above, it may seem that the quantum computer was in state `01` all along and we just weren't aware of it, so that 36% and 64% were just representing our uncertainty. A key point of quantum mechanics, however, is that we really can be in a distinct state that had a 36% chance of being measured as `01` and 64% chance of being measured as `11`, and that it was the act of measuring that pushed it to the state that is always `01`.

With the probabilities described earlier, it seems that we were "more strongly" in state `11` than in state `01`, as the former had a higher probability of being measured. In quantum computing, we would say that the state `11` had a larger *amplitude* than the state `01`. The amplitude of a state is not merely its probability of measurement, however. Rather, it is a *complex* number associated with the state. You can obtain the probability of measuring a state by squaring the absolute value of its amplitude (the absolute value of $a + bi$ is $|a + bi| = \sqrt{a^2 + b^2}$), so that if the amplitude of a state is z, the probability of measuring that state is $|z|^2$. So if state `01` has a 36% chance of being measured, its amplitude could be 0.6, or $0.6i$, or $\sqrt{0.3}+i\sqrt{0.06}$, or infinitely many other values. While each of these amplitudes produce an equal probability of being measured, they are affected differently by quantum operations, which is partly why there is a difference between being in a superposition of state `01` and `11` and not knowing whether we are in state `01` or state `11`.

We can now come up with a mathematical representation of the quantum state: we can view it as a vector of amplitudes by enumerating all possible states (e.g. `00`, `01`, `10` and `11` in a 2 qubit system) and putting the amplitude of each state in a vector. Fig. 1 shows a possible vector representation of the situation we have been discussing. Because the vector contains all possible combinations of bits, an n-qubit system will have an amplitude vector with 2^n elements.

The fact that the square of the absolute value of the amplitude corresponds to the probability of measurement of a state has an important consequence: the sum of the probabilities of measuring each state must be 1, so if v is an amplitude vector, we must have $\sum_i |v_i|^2 = 1$. We recognize the left hand side as the square of the standard complex vector norm, so we must have $\|v\| = 1$, i.e. any amplitude vector is a (complex) *unit* vector.

How do we manipulate a quantum state? A fundamental postulate of quantum mechanics is that the evolution of any quantum state can be represented by a *unitary* transformation of that state's amplitude vector [8]. Hence, any operation that we might take on state v will lead to a new state Uv, where U is a unitary matrix. This is consistent with our observation that any amplitude vector has to be a unit vector: we would hope that Uv is also a valid amplitude vector, i.e. that it is also a unit vector, and if U is a unitary matrix, then $\|Uv\| = \|v\|$ for all v, so this criteria is met. Indeed, it can be shown that unitary matrices not only preserve norm, but that all norm-preserving matrices are unitary [4], so if the evolution of a quantum state has to be a linear transformation (which is out of the scope of this project to prove), it makes sense that it would be described by a unitary matrix.

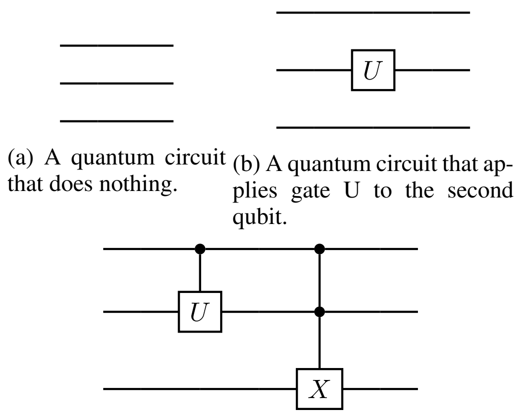
3. Quantum Gates and Quantum Circuits

While we can represent any quantum algorithm as a single unitary matrix, this representation is not always practical. In particular, it is not obvious how to implement an arbitrary matrix in hardware: a qubit is usually a small particle like a photon or an atom, and while it is possible to manipulate one qubit at a time or sometimes two qubits at once, it is often not clear how to make dozens of qubits interact in the highly specific way represented by a given matrix.

To address this problem, we introduce *quantum gates*. These are simple quantum operations that can be chained to produce more complex operations, similar to logic gates in classical computing. Unlike large quantum algorithms which can act on many qubits at once in arbitrary ways, quantum gates can either act on one qubit at a time or can only act on multiple qubits in specific ways. Since quantum gates are themselves quantum operations, they have a unitary matrix representation, and chaining of quantum gates corresponds to multiplying those matrices.

If a set of quantum gates can be used to implement any quantum algorithm, it is called universal. Our goal in this paper is to define a universal set of gates and show how any unitary matrix can be decomposed into that set.

We often represent quantum gates using quantum circuit notation. Fig. 2a shows the simplest quantum circuit with 3 qubits: one which does nothing. The horizontal axis shows the passage of time, and each line represents a qubit.



(a) A quantum circuit that does nothing.

(b) A quantum circuit that applies gate U to the second qubit.

(c) A circuit with controlled U and X gates.

Figure 2: Some simple quantum circuits.

Since time flows from left to right, but when multiplying $U_n \cdots U_1 v$, the rightmost matrix is applied first to the input v, the order of gates is reversed when going between the circuit and matrix representations. In Fig. 2a, the lines stay unmodified throughout, and so the qubits are unaltered by the end of the circuit.

3.1. Single Qubit Gates

The simplest type of quantum gate is a single qubit gate, which corresponds to an operation that only affects a particular qubit. Fig 2b shows a quantum circuit in which the gate U is applied to the second qubit. Being a single-qubit operation, it must be possible to represent U as a 2×2 unitary matrix, $U = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$. However, when we apply U to the whole 3-qubit system, we must make it so that U is applied to the second qubit in all places, giving us:

$$U = \begin{array}{cccccccc|c} \texttt{000} & \texttt{001} & \texttt{010} & \texttt{011} & \texttt{100} & \texttt{101} & \texttt{110} & \texttt{111} & \\ \mathbf{a} & 0 & \mathbf{b} & 0 & 0 & 0 & 0 & 0 & \texttt{000} \\ 0 & \mathbf{a} & 0 & \mathbf{b} & 0 & 0 & 0 & 0 & \texttt{001} \\ \mathbf{c} & 0 & \mathbf{d} & 0 & 0 & 0 & 0 & 0 & \texttt{010} \\ 0 & \mathbf{c} & 0 & \mathbf{d} & 0 & 0 & 0 & 0 & \texttt{011} \\ 0 & 0 & 0 & 0 & \mathbf{a} & 0 & \mathbf{b} & 0 & \texttt{100} \\ 0 & 0 & 0 & 0 & 0 & \mathbf{a} & 0 & \mathbf{b} & \texttt{101} \\ 0 & 0 & 0 & 0 & \mathbf{c} & 0 & \mathbf{d} & 0 & \texttt{110} \\ 0 & 0 & 0 & 0 & 0 & \mathbf{c} & 0 & \mathbf{d} & \texttt{111} \end{array}$$

Notice how the non-zero entries are precisely those where either no qubit has changed between the rows and the columns, or only the second qubit has changed: this is because our gate only applies to the second qubit.

An important single-qubit gate is the X gate, $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, also called the NOT gate because if the amplitude of the qubit is fully on 0 or 1, then it flips it to 1 or 0, respectively. Other important single-qubit gates [7] are the rotation around y, $R_y(\theta) = \begin{bmatrix} \cos\theta/2 & \sin\theta/2 \\ -\sin\theta/2 & \cos\theta/2 \end{bmatrix}$, the rotation around z, $R_z(\alpha) = \begin{bmatrix} e^{i\alpha/2} & 0 \\ 0 & e^{-i\alpha/2} \end{bmatrix}$ and the phase-shift gate $Ph(\phi) = \begin{bmatrix} e^{i\phi} & 0 \\ 0 & e^{i\phi} \end{bmatrix}$. We will see in Sec. 4.4 that the phase-shift and rotation gates can be used to represent any single-qubit gate, so they will be part of our universal set.

3.2. Controlled Gates

The only type of multi-qubit gates we will consider are the *controlled* gates. A controlled gate is the same as a single-qubit gate in that it only affects a single bit, but whether it affects that bit at all is controlled by the value of another bit. In Fig. 2c, we see a U gate being applied to the second bit with the first bit as control. This means that the U gate will only be applied to the second bit in states where the first bit is 1, whereas in states where the first bit is 0, the second bit will not change. In matrix terms, this corresponds to:

$$U = \begin{array}{cccccccc|c} \texttt{000} & \texttt{001} & \texttt{010} & \texttt{011} & \texttt{100} & \texttt{101} & \texttt{110} & \texttt{111} & \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \texttt{000} \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & \texttt{001} \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & \texttt{010} \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & \texttt{011} \\ 0 & 0 & 0 & 0 & \mathbf{a} & 0 & \mathbf{b} & 0 & \texttt{100} \\ 0 & 0 & 0 & 0 & 0 & \mathbf{a} & 0 & \mathbf{b} & \texttt{101} \\ 0 & 0 & 0 & 0 & \mathbf{c} & 0 & \mathbf{d} & 0 & \texttt{110} \\ 0 & 0 & 0 & 0 & 0 & \mathbf{c} & 0 & \mathbf{d} & \texttt{111} \end{array}$$

Notice that the corner in which the first bit is 0 is an identity matrix, and the structure from the earlier matrix U is only present where the first qubit is 1.

It is possible to have multiple control bits, which will further increase the number of places in which the U gate is replaced with the identity matrix. The second gate in Fig. 2c is shown with two control bits.

While the target gate can be anything, our a goal in this paper will be to restrict our usage of controlled gates to only the controlled-X gate with a *single* control bit. This gate is often called the controlled-NOT, or CNOT gate, and is the only multi-qubit gate included in our universal set.

4. Quantum Circuit Decomposition

We have seen that any quantum algorithm can be represented as a unitary matrix, but that practical implementations are restricted to a particular set of quantum gates. We must therefore find a way to decompose any $2^n \times 2^n$ unitary matrix into a product of simple quantum gates.

We will proceed as follows: first, we will use QR decomposition with the Givens operation to decompose the unitary matrix into a product of simpler unitary matrices. Then,

$$U = \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \to G_{31}U = \begin{bmatrix} * & * & * \\ * & * & * \\ 0 & * & * \end{bmatrix} \to$$

$$G_{21}G_{31}U = \begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & * & * \end{bmatrix} \to G_{32}G_{21}G_{31}U = \begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{bmatrix}$$

Figure 3: QR decomposition by Givens operations in the 3×3 case. The $*$ symbol represents a possibly non-zero value, the next subvector to have its bottom entry zeroed out is highlighted.

we will show that these simpler matrices can be decomposed into a circuit of single-qubit gates and multi-qubit controlled gates. Then we will show that any multi-qubit controlled gate can be decomposed into a circuit of gates controlled by a single qubit. Finally, we will show that any single-qubit gate can be decomposed into a circuit of phase-shift and rotation gates, and that any gate controlled by a single qubit can be decomposed into a circuit of single-qubit gates and CNOT gates, leaving us with a universal set consisting only of phase-shift, rotation, and CNOT gates.

4.1. QR Decomposition with Givens Operations

The QR factorization factorizes a full rank square matrix into $U = QR$ where Q is orthogonal and R is upper triangular. In practice, we can build Q by successively zeroing out entries of U until the whole lower triangle is zeroed out, producing R. In our case, each zeroing iteration will be produced by a matrix that Cybenko calls a "quantum Givens operation", which is the generalization of the Givens rotation to the complex numbers. The Givens operations will be designed to be unitary, and R will also be unitary: $U = QR$, so $R = Q^*U$, where both Q^* and U are unitary, so R is the product of two unitaries, and therefore unitary.[1]

To perform the QR factorization, we successively premultiply U by a matrix G_{ij} that will zero out entry (i, j), in the order illustrated in Fig. 3 (from bottom to top – up to the diagonal – and left to right). We want G_{ij} to have the following block form:

$$G_{ij} = \begin{bmatrix} \mathbf{I} & 0 & 0 \\ 0 & \mathbf{G} & 0 \\ 0 & 0 & \mathbf{I} \end{bmatrix},$$

where G is a 2×2 matrix placed in such a way that it will multiply the vector $\begin{bmatrix} u_{i-1,j} \\ u_{ij} \end{bmatrix}$ (i.e. the entry we want to zero out as well as the one above it) when we do the multiplication $G_{ij}U$. In Fig. 3, at each step the section of the matrix we want to affect with our next G is highlighted (G also multiplies other sections, but we will show that this never unzeros out previously zeroed entries).

If the target subvector is of the form $v = \begin{bmatrix} a \\ b \end{bmatrix}$, we want our matrix G to be such that

$$Gv = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sqrt{|a|^2 + |b|^2} \\ 0 \end{bmatrix}.$$

The lower element of Gv is zero because we are trying to zero out that entry, while the top element is $\sqrt{|a|^2 + |b|^2}$ because we need G to be unitary. As we saw in Sec. 2, unitary matrices are norm-preserving, so we must have $\|Gv\| = \|v\| = \sqrt{|a|^2 + |b|^2}$. Cybenko proposes $G = \frac{1}{\sqrt{|a|^2+|b|^2}} \begin{bmatrix} \overline{a} & \overline{b} \\ b & -a \end{bmatrix}$, which meets our our requirements of being unitary ($GG^* = I$) and taking v to the required Gv, as can be verified by matrix multiplication.

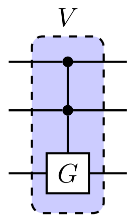
Thus each matrix G_{ij} is unitary (because the G block is unitary) and will zero out the (i, j)th entry. It remains to be shown that, if we zero out entries in the order specified earlier, we will never unzero previously zeroed entries. We proceed by induction starting from the case where we have successfully zeroed out the bottom left entry, and show that at each subsequent step, we are not unzeroing any previously intentionally zeroed entry. Suppose the matrix is in a valid intermediate state and entry (i, j) has been zeroed out, then the column containing (i, j) is of the form $(*, \ldots, \boxed{*, 0, 0}, \ldots)$, where the $*$ are possibly nonzero entries and the first zero is entry (i, j). Now, suppose we are multiplying by a Givens matrix G_{kl}. There are three possibilities for the row of G_{kl} that will be multiplied with our column to give the new (i, j)th entry: a) $(0, \ldots, \boxed{0, 1, 0}, \ldots)$, b) $(0, \ldots, \boxed{*, *, 0}, \ldots)$, or c) $(0, \ldots, \boxed{0, *, *}, 0, \ldots)$, where in each case the boxed entries match the ones from the column. Since all other entries of the row are zero, we only need to consider the dot product of the boxed entries. In cases a) and c), the result will be 0 since the non-zero entries in the row match zero entries in the column, and vice-versa. Case b) is where order comes in: the dot product can only be non-zero if the $*$ entry in the column is non-zero. This only happens if we are attempting to zero out an entry in row i. However, our ordering is such that if we ever attempt to zero out entry (i, l) where $l > j$, then we must have zeroed out the entry $(i - 1, j)$ beforehand (if we hadn't, then $(i - 1, j)$ is on the diagonal, but then our ordering is such that we will never zero out entries in row i again), so the $*$ entry in the boxed section will always be 0 in case b), and the result will also be 0.

Thus for any unitary U, we can write $G_{n,n-1} \ldots G_{n1}U = R$ where the $G_{\ldots}$ as well as R are unitary. This gives us the QR decomposition $U = QR$ where $Q = G^*_{n1} \ldots G^*_{n,n-1}$.

[1]See Module 8 video "Unitary Matrices."

$$\begin{array}{cccccccc|c}
000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 & \\
\mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 000 \\
0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 001 \\
0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 010 \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 011 \\
0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 100 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 101 \\
0 & 0 & 0 & 0 & 0 & 0 & \mathbf{a} & \mathbf{b} & 110 \\
0 & 0 & 0 & 0 & 0 & 0 & \mathbf{c} & \mathbf{d} & 111
\end{array}$$

(a) Matrix representation.



(b) Quantum circuit.

Figure 4: A simple Givens operation where G covers only one the last qubit and all other qubits are 1.

4.2. Decomposition into Multi-Bit Controlled Gates

We now have a decomposition of U into simpler matrices, which we must further decompose into valid quantum gates. For now, we consider all single-qubit gates as well as controlled gates controlled by any number of qubits to be valid gates. Later sections will further restrict this set.

4.2.1. Decomposition of Q

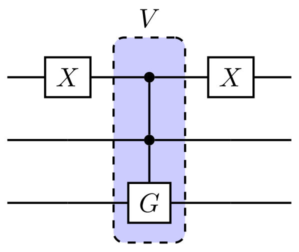
In our QR decomposition, Q can be written as a product of inverses of Givens matrices. An inverse Givens matrix follows the same block form as the original matrix but with G^* taking the place of G, so it is also a Givens matrix.

Are Givens matrices already valid quantum gates? Unfortunately, that depends on the location of the G block. In the ideal case, the G block is found all the way at the bottom right of the matrix (Fig. 4a). In that case, we can tell from Sec. 3.2 that it follows the form of a controlled G gate on the last qubit, with all other qubits as controls. The corresponding circuit form is shown in Fig. 4b. We will use V to refer to this ideal configuration of a Givens matrix.

Now, suppose we have a Givens matrix W, which has its G block in a different location than the bottom right. Our strategy to apply W will be to move the G block in V to its location in W using valid quantum gates (our procedure might require that V have a different ordering of the elements in its G block than W, but the appropriate ordering will be easy to find). We can divide this into two steps: swapping the last two columns of V with the columns of W containing its G block, and swapping the last two rows of V with the rows of W containing the G block. Swapping the rows of a matrix can be done by multiplying on the left with a *permutation matrix*, and that multiplying on the right will swap the columns. Indeed, since the G block of W is always on the diagonal, the indices of the target rows and columns are the same, so we can use the same

$$\begin{array}{cccccccc|c}
000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 & \\
\mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 000 \\
0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 001 \\
0 & 0 & \mathbf{a} & \mathbf{b} & 0 & 0 & 0 & 0 & 010 \\
0 & 0 & \mathbf{c} & \mathbf{d} & 0 & 0 & 0 & 0 & 011 \\
0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 100 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 101 \\
0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 110 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 111
\end{array}$$

(a) Matrix representation.



(b) The corresponding quantum circuit.

Figure 5: Case 1: A Givens operation where G only covers the last qubit flipping, but the first qubit is 0.

permutation matrix on both sides.

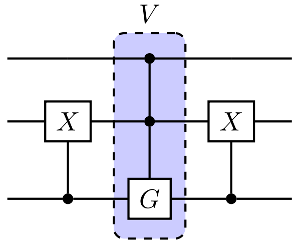
We are thus looking for a permutation P such that $W = PVP$. However, not every single permutation is a valid quantum gate, so we will have to build up P using a circuit of valid gates. To do so, we will exclusively use the X gate (both in controlled and uncontrolled form), so that we can write $W = X_1 \dots X_n V X_n \dots X_1$. It is easily shown that any X_i gate is its own inverse, so we can also write $V = X_n \dots X_1 W X_1 \dots X_1$. Solving this latter problem is more intuitive to explain, so this is the approach we will take here (i.e. rather going from matrix V to W, we will show how to take any Givens matrix W one step closer to V), and we can later implement the $W = X_1 \dots X_n V X_n \dots X_1$ circuit by reversing the order of the circuit on each side.

There are two cases to consider when taking W to V. In case 1, the G block is in a location where one bit is always 0 (Fig. 5a). As we can see in Fig. 4a, in V, the G block overlaps only places where every bit except the last one is 1. Thus we want a permutation matrix that permutes the rows/columns where the offending bit is 0 with the rows/columns where that bit is 1. This is exactly what an uncontrolled X gate on the offending bit does, and it is thus the gate we use here (Fig. 5b).

In case 2, the G block spans a bit flip that is not the last bit (Fig. 6a). As we have seen, in V, only the last bit flips. In this case, we want to swap entries of W so that all the entries of G move to the same value of the offending bit (if we happen to move G to a location where the offending bit

$$\begin{array}{cccccccc|c}
000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 & \\
\mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 000 \\
0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 001 \\
0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 010 \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 011 \\
0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 100 \\
0 & 0 & 0 & 0 & 0 & \mathbf{a} & \mathbf{b} & 0 & 101 \\
0 & 0 & 0 & 0 & 0 & \mathbf{c} & \mathbf{d} & 0 & 110 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 111
\end{array}$$

(a) Matrix representation.



(b) The corresponding quantum circuit.

Figure 6: Case 2: A Givens operation where G covers both the second and third qubit flipping.

is 0, we can fix this using case 1). To do so, we observe that the last bit will always also flip across G, so we can use it as a pivot of sorts by using a controlled X gate on the offending bit, controlled by the last bit. This will move all entries of G corresponding to one value of the offending bit to the other value, so that G no longer spans that bit swap.

With this procedure, any offending bit can be made non-offending with at most one application of case 2 and case 1, so we can take any Givens matrix W to V by following those two cases until no offending bits are left. Reversing the permutation circuit on each side allows us to implement W using V, as depicted in Fig. 5b and 6b.

4.2.2. Decomposition of R

Since R is unitary and upper triangular, it must be diagonal, which we prove now: R's diagonal entries are non-zero (see, e.g. [1] Theorem 4.11), and since R is unitary, $RR^* = I$, so the non-diagonal entries of RR^* are zero. Now, consider entry $(1, n)$ of RR^*. Its value is given by $\sum_{k=1} r_{1k}\overline{r_{nk}}$, but since R is upper triangular, the entries r_{nk} are 0 for all $k < n$, so only r_{nn} is non-zero, making the $(1, n)$th entry of RR^* equal to $r_{1n}\overline{r_{nn}}$. Now, r_{nn} is non-zero since it is on the diagonal, and unless $n = 1$ (in which case R is trivially diagonal), the $(1, n)$th entry of RR^* is 0, so we must have $r_{1n} = 0$. Now that we know this, the same argument shows that the $(1, n-1)$th entry of RR^* is equal to $r_{1,n-1}\overline{r_{n-1,n-1}}$, and since $r_{n-1,n-1} \neq 0$, we must have $r_{1,n-1} = 0$ if $n - 1 \neq 1$. Continuing this argument until $(1, 2)$, we find that only entry $(1, 1)$ is non-zero in the first row. The same argument can be used to show that only entry $(2, 2)$ is non-zero in the second row, and so on, showing that R is diagonal. Since R is diagonal, we can use the method in the previous section to decompose R into quantum gates by targeting each 2×2 block on the diagonal in turn as if it was a G matrix.

4.3. Reducing Multi-Qubit Controlled Gates

We have now converted our unitary matrix into a quantum circuit of well-defined quantum gates. Here, we reduce controlled gates with multiple control bits to gates with at most one control bit.

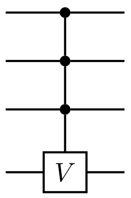
We take a recursive approach: if a gate has $n > 1$ control bits, we will show that we can decompose it into a circuit of gates with at most $n - 1$ control bits. If $n - 1 > 1$, we can use the same method to get gates with at most $n - 2$ control bits, and so on until all gates have at most one control bit.

This reduction requires a key observation: suppose we are trying to decompose a multi-control bit gate V (e.g. Fig 7a). Since V is a unitary matrix, it must have a square root W, i.e. a matrix such that $V = W^2$. We show this by observing that since V is unitary, it can be written as $V = PDP^*$ (Noble and Daniel (8.6)), where D is diagonal and P is unitary. If we take D' to be a matrix with square roots of the entries of D on its diagonal (all of which exist, since we are working with complex numbers), then $D = D'^2$, so we can write $W = PD'P^*$, and so $WW = PD'P^*PD'P^* = PD'^2P^* = PDP^* = V$.

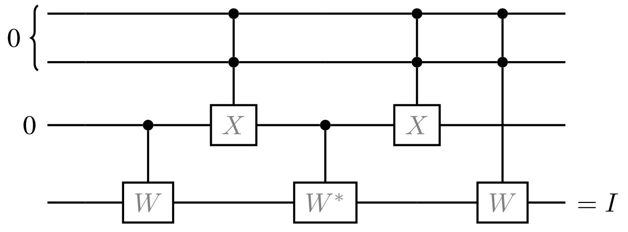
W is unitary (and can thus be used as a gate): from the diagonalization of V and the fact that it is unitary, we have $VV^* = PDP^*PD^*P^* = PDD^*P^* = I$, which we can rearrange into $DD^* = I$. Since D is diagonal, the entries δ_{ii} of D are such that $\delta_{ii}\overline{\delta_{ii}} = |\delta_{ii}| = 1$. This implies that the entries δ'_{ii} of D' are also such that $|\delta'_{ii}| = 1$ since $1 = |\delta_{ii}| = |\delta'^2_{ii}| = |\delta'_{ii}|^2$. Thus $D'D'^* = I$, so $W = PD'P^*$ is a product of unitary matrices and therefore unitary.

We now show how to take an n-bit controlled gate (Fig. 7a) to a circuit of gates with at most $n - 1$ control bits. We create a circuit such that, if the n control bits are all 1, we will apply W twice, which is V since $WW = V$, but if any of the control bits are 0, we will apply W and W^*, which does nothing since $WW^* = W^*W = I$. The control bits are numbered $1, 2, \ldots, n$, and bit n will be our "special" control bit. We follow the following steps (see Fig. 7):

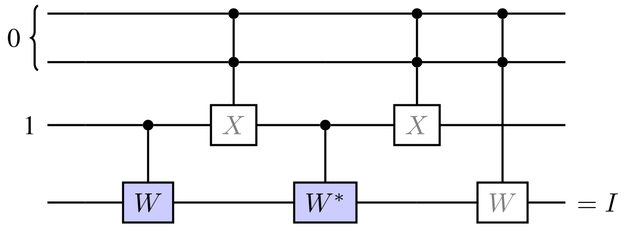
1. Apply W controlled with bit n to the target bit.
2. Apply X controlled with bits $1, \ldots, n-1$ to bit n.
3. Apply W^* controlled with bit n to the target bit.
4. Apply X controlled with bits $1, \ldots, n-1$ to bit n (this brings bit n back to its original state).
5. Apply W controlled with bits $1, \ldots, n-1$ (*excluding* bit n) to the target bit.



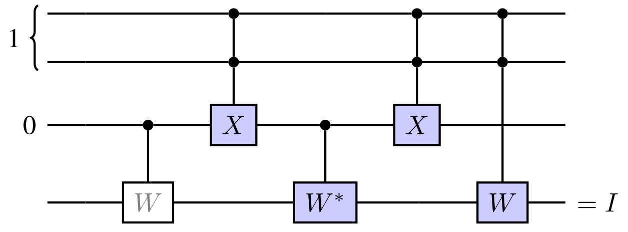
(a) The original controlled gate V.



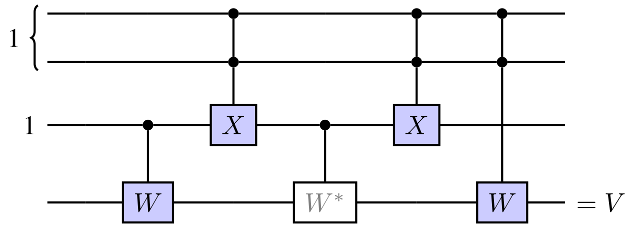
(b) Bits $1, \ldots, n-1$ contain a 0, bit n is 0.



(c) Bits $1, \ldots, n-1$ contain a 0, bit n is 1.



(d) Bits $1, \ldots, n-1$ are 1, bit n is 0.



(e) Bits $1, \ldots, n$ are 1.

Figure 7: The pathways taken in different cases when reducing the number of control bits. In each case, the highlighted gates are the ones that are active due to the control bits, while the inactive gates are grayed out.

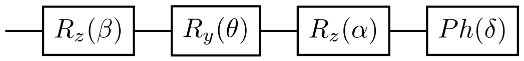


Figure 8: Decomposition of any single qubit gate.

To see how this works, consider the four possible cases depending on whether bit n is 0 or 1 and bits $0, \ldots, n-1$ are all 1s or contain a 0 (step 4 always brings bit n back to its initial state, so we don't explain it each time):

- Bit n is 0 and bits $0, \ldots, n-1$ contain a 0 (Fig. 7b): we don't apply W at step 1. Bit n is unchanged at step 2, so we don't apply W^* at step 3, and we don't apply W at step 5. Thus nothing happens to the target.
- Bit n is 0 but bits $0, \ldots, n-1$ are all 1 (Fig. 7c): we don't apply W at step 1. Bit n becomes 1 at step 2, so we apply W^* at step 3. At step 5, we apply W. The net result is to apply $WW^* = I$, which does nothing.
- Bit n is 1 but bits $0, \ldots, n-1$ contain a 0 (Fig. 7d): we apply W at step 1. Bit n doesn't change at step 2, so we apply W^* at step 3. We don't apply W at step 5. The result is $W^*W = I$, which does nothing.
- All bits $1, \ldots, n$ are 1 (Fig. 7e): we apply W at step 1. Bit n becomes 0 at step 2, so we don't apply W^* at step 3. At step 5, we apply W. The net result is to apply $WW = V$ to the target.

Applying this approach recursively, we can reduce any multi-controlled gate to a circuit of single-controlled gates.

4.4. Reducing Single-Qubit Gates

We now show how to decompose any 2×2 unitary matrix into a product of phase-shift and rotation gates. First, note that any 2×2 unitary can be written in the following form[2]:

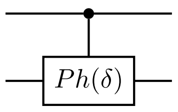
$$\begin{bmatrix} e^{i(\delta+\alpha/2+\beta/2)} \cos\theta/2 & e^{i(\delta+\alpha/2-\beta/2)} \sin\theta/2 \\ -e^{i(\delta-\alpha/2+\beta/2)} \sin\theta/2 & e^{i(\delta-\alpha/2-\beta/2)} \cos\theta/2 \end{bmatrix}.$$

This can easily be rewritten as:

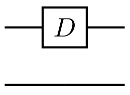
$$\overbrace{\begin{bmatrix} e^{i\delta} & 0 \\ 0 & e^{i\delta} \end{bmatrix}}^{\text{Part 1}} \overbrace{\begin{bmatrix} e^{i\alpha/2} & 0 \\ 0 & e^{-i\alpha/2} \end{bmatrix} \begin{bmatrix} \cos\theta/2 & \sin\theta/2 \\ -\sin\theta/2 & \cos\theta/2 \end{bmatrix} \begin{bmatrix} e^{i\beta/2} & 0 \\ 0 & e^{-i\beta/2} \end{bmatrix}}^{\text{Part 2}} \tag{1}$$

The distinction between Part 1 and Part 2 matrices will be relevant in the next section. For now, we recognize the first matrix as being a phase-shift gate $Ph(\delta)$, the second and last matrix as being rotations around z, $R_z(\alpha)$ and $R_z(\beta)$, and the third matrix as being a rotation about y, $R_y(\theta)$, thus showing that any single-qubit gate can be implemented as a combination of phase-shift and rotation gates (Fig. 8).

[2] A proof of this can be found in [9]. I do not reproduce it here because it involves more complex arithmetic than linear algebra and is stated as a brute fact by both Cybenko and Barenco et al.

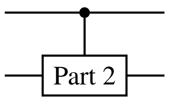


(a) Controlled version.

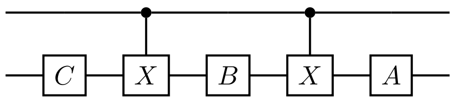


(b) Single-qubit version.

Figure 9: Equivalent circuits for the controlled phase-shift gate, where D is defined in Sec. 4.5.



(a) Controlled version.



(b) Single-qubit and CNOT version.[3]

Figure 10: Equivalent circuits for the Part 2 gates, where A, B and C are defined in Sec. 4.5.

4.5. Reducing 2-Qubit Controlled Gates

We are still using arbitrary targets in our controlled gates. We now want to restrict our gate set so that the only 2-qubit gate we use is the CNOT gate, i.e. the controlled X gate. To do so, we implement Part 1 and Part 2 of the factorization found earlier using only CNOT and single-qubit gates.

Remarkably, the controlled version of the Part 1 matrix, $Ph(\delta)$, can be reduced to a single qubit gate on the control bit, namely $D = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\delta} \end{bmatrix}$. To see this, create the 4×4 matrix corresponding to the controlled circuit in Fig. 9a and to the single-qubit gate circuit in Fig. 9b following the procedure from Sec. 3.1 and 3.2. In both cases, the resulting matrix is $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{i\delta} & 0 \\ 0 & 0 & 0 & e^{i\delta} \end{bmatrix}$.

To implement the controlled version of Part 2 with only CNOT gates, Cybenko proposes the following matrices:

$$A = \begin{bmatrix} e^{i\alpha/2} & 0 \\ 0 & e^{-i\alpha/2} \end{bmatrix} \begin{bmatrix} \cos\theta/4 & \sin\theta/4 \\ -\sin\theta/4 & \cos\theta/4 \end{bmatrix}$$

$$B = \begin{bmatrix} \cos-\theta/4 & \sin-\theta/4 \\ -\sin-\theta/4 & \cos-\theta/4 \end{bmatrix} \begin{bmatrix} e^{-i(\alpha+\beta)/4} & 0 \\ 0 & e^{i(\alpha+\beta)/4} \end{bmatrix}$$

$$C = \begin{bmatrix} e^{-i(\alpha-\beta)/4} & 0 \\ 0 & e^{i(\alpha-\beta)/4} \end{bmatrix}$$

These matrices have a remarkable property: $ABC = I$ and $AXBXC$ is equal to the product of matrices in Part 2. This means that to get a controlled version of Part 2, we can interleave the single-qubit gates A, B and C with CNOT gates, so that, if the control bit is 1, $AXBXC$ = Part 2 will be applied, and otherwise $ABC = I$ will be (Fig. 10).

5. Conclusion

We have thus managed to factor any quantum algorithm (represented by a unitary matrix) into a circuit of elementary quantum gates, namely the phase-shift, rotation, and CNOT gates, which is the most widely used set of gates in current quantum circuits [10]. The very fact that we were able to do this has important consequences, among which:

- We have proven that the set phase-shift + rotation + CNOT is a universal gate set.
- Any hardware manufacturer that implements these gates above can be confident that they have built a universal quantum computer.
- Not only does any possible computation correspond to a unitary matrix (which we assumed in Sec 2), but the converse (which we did not assume) is also true: any unitary matrix corresponds to a possible quantum computation, implementable with our gates.

What of the practical uses of the algorithm itself? Some researchers have called quantum circuit decomposition the quantum equivalent of compilation [11], which seems to imply that people will often design algorithms as unitary matrices and compile them to gate form. If this becomes the norm, then our algorithm may be very useful indeed. Sadly, there is a good reason to believe that large quantum algorithms will rarely be designed directly in matrix form: the matrices are huge ($2^n \times 2^n$)! For example, assuming 8 bytes per entry, the matrix for a 16-qubit algorithm would occupy 32GB of storage. Indeed, arguably the whole reason quantum computing is so powerful is that it allows us to perform a $2^n \times 2^n$ matrix multiplication with just n qubits.

Though we may rarely compile entire algorithms from their matrix form, new multi-bit gates are bound to be invented from time to time. When this happens, we will need to implement them in hardware, and the algorithm described here gives us the means to do so. In that sense, quantum circuit decomposition is crucial to the progress of quantum computing.

There is still substantial research aiming to improve both the number of elementary gates in the final decomposition [12–14] and the speed at which the decomposition is computed [11]. The algorithm presented here shines by its relative simplicity, but not by its optimality in either of

[3]Cybenko shows the gates in the opposite order in the quantum circuit ($A - B - C$ instead of $C - B - A$), but this struck me as incorrect since quantum circuits proceed from left to right unlike matrix multiplication which goes from right to left. Cybenko is more of an expert than me, and thus more likely to be correct, but I chose to go with my understanding in my version of the diagram.

these dimensions. Indeed, there are further aspects that our method ignores entirely. For instance, while our method produces a decomposition without using extra qubits, it also couldn't use extra qubits even if they were available. Barenco et al. [7] show that by taking advantage working qubits, it is possible to dramatically reduce the number of gates used in a quantum circuit decomposition. Another question not addressed here is that of other universal sets of gates: while our set is a popular one, it is far from the only universal gate set [10]. Indeed, it is possible to create universal gate sets containing a single gate type, rather than the three types we used here [15].

There is thus much in the quantum circuit decomposition literature that I couldn't cover here, but the method I presented proves important facts about quantum computation and hopefully serves as a good introduction to the field.

References

[1] Thomas S Shores. *Applied linear algebra and matrix analysis*. Undergraduate Texts in Mathematics. Springer International Publishing, Basel, Switzerland, 2 edition, May 2018.

[2] George Cybenko. Reducing quantum computations to elementary unitary operations. *Computing in Science & Engineering*, 3(2):27–32, 2001.

[3] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[4] Andy Matuschak and Michael A. Nielsen. Quantum computing for the very curious. 2019. `https://quantum.country/qcvc`.

[5] Andy Matuschak and Michael A. Nielsen. How does the quantum search algorithm work? `https://quantum.country/search`, 2019.

[6] John Preskill. Simulating quantum field theory with a quantum computer. *arXiv preprint arXiv:1811.10085*, 2018.

[7] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical review A*, 52(5): 3457, 1995.

[8] Andy Matuschak and Michael A. Nielsen. Quantum mechanics distilled. `https://quantum.country/qm`, 2020.

[9] gls. General parametrisation of an arbitrary 2×2 unitary matrix? `https://quantumcomputing.stackexchange.com/questions/5199/general-parametrisation-of-an-arbitrary-2-times-2-unitary-matrix`, 2019.

[10] Colin P. Williams. Quantum gates. In *Texts in Computer Science*, pages 51–122. Springer London, 2011. doi: 10.1007/978-1-84628-887-6_2. URL `https://doi.org/10.1007/978-1-84628-887-6_2`.

[11] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, and Cyril Allouche. Quantum circuits synthesis using householder transformations. *Computer Physics Communications*, 248:107001, 2020.

[12] Michal Sedlák and Martin Plesch. Towards optimization of quantum circuits. *Open Physics*, 6(1):128–134, 2008.

[13] Juha J Vartiainen, Mikko Möttönen, and Martti M Salomaa. Efficient decomposition of quantum gates. *Physical review letters*, 92(17):177902, 2004.

[14] Vivek V Shende, Stephen S Bullock, and Igor L Markov. Synthesis of quantum-logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6):1000–1010, 2006.

[15] David Elieser Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 425(1868):73–90, 1989.