

Florida State University Libraries

Honors Theses

The Division of Undergraduate Studies

2012

A Quantum Compiler for Topological Quantum Computation

Caitlin Carnahan



Abstract

A quantum computer is a device that exploits the strange properties of quantum mechanics in order to perform computations that are not feasible on a classical computer. To implement a quantum computer, it will be necessary to maintain the delicate quantum superpositions formed during computation; this is a very difficult problem because quantum systems, by their very nature, are incredibly fragile. However, it is possible to implement a finite set of quantum gates to the required accuracy, which makes it possible to perform *fault-tolerant* quantum computing, a scheme that minimizes error propagation in computations. The problem then becomes developing a method to build arbitrary quantum operations using this finite set of fault-tolerant gates. This can be accomplished by using the Solovay-Kitaev theorem, which proves that any unitary operation can not only be simulated, but done so efficiently to within a small margin of approximation using only the gates in the universal fault-tolerant gate set. The purpose of this research is to create an efficient program that demonstrates the process of the Solovay-Kitaev theorem using various universal gate sets. Essentially, the program presented in this paper translates a desired operation into the “machine code” of a quantum computer and therefore acts as a “quantum compiler”. This project focuses specifically on topological quantum computing in which the fault-tolerant gate set can be visualized as elementary braids formed by worldlines traced out by exotic quasiparticles known as Fibonacci anyons.

Keywords: Quantum Computation, Fibonacci Anyons, Solovay-Kitaev Theorem

THE FLORIDA STATE UNIVERSITY
COLLEGE OF ARTS AND SCIENCES

A QUANTUM COMPILER FOR TOPOLOGICAL QUANTUM
COMPUTATION

By

CAITLIN CARNAHAN

A Thesis submitted to the
Department of Physics
in partial fulfillment of the requirements for graduation with
Honors in the Major

Degree Awarded:
Spring, 2012

The members of the Defense Committee approve the thesis of Caitlin Carnahan defended on April 13, 2012.

Dr. Nicholas Bonesteel
Thesis Director

Dr. Robert van Engelen
Outside Committee Member

Dr. Irinel Chiorescu
Committee Member

Contents

1	Introduction	2
2	Quantum Computing Basics	4
2.1	The Qubit	4
2.2	Quantum Gates and Quantum Circuits	6
2.2.1	Quantum Gates	6
2.2.2	Quantum Circuits	9
2.2.3	Two-Qubit Gates and Universal Gate Sets	10
3	The Solovay-Kitaev Theorem	13
3.1	The Theorem	13
3.2	The Details	14
4	Fibonacci Anyons	19
4.1	Braiding	19
4.2	Weaving	21
5	The Program	23
5.1	Introduction	23
5.2	Environment	23
5.2.1	Fibonacci Anyon Rotations	24
5.2.2	Other Universal Rotations	27
5.3	Results	28
5.4	Applications and Future Work	31
6	References	33

1 Introduction

Two of the greatest intellectual achievements of the 20th century were the development of the theory of computation and information, from which revolutionary technology like the personal computer has stemmed, and the discovery of quantum theory, which explains the behavior of matter at and below the atomic level. It was, in fact, the understanding of quantum theory as applied to semiconductors that led to the development of the fundamental component of the classical computer, the transistor. Despite all of the complexity of today's modern computers, the transistors that they are composed of are only "switches" - they can either be on or off. Essentially, they are no different than a bead in an abacus. While modern computers are extremely complex and sophisticated in terms of the amount of signals they utilize and the methods in which they control those signals, their operation is fundamentally no different than that of an abacus. They are both *classical* computers.

A new paradigm which utilizes the strange properties of quantum mechanics may allow us to create computing devices beyond the scope of classical computing. These new devices, known as quantum computers, manipulate information in a fundamentally different way. They would be able to perform currently feasible computations much faster and solve problems for which there is no known efficient solution at the moment. An example of the former is Grover's search algorithm[1], which shows that a quantum computer can search an unordered list of N elements in $O(\sqrt{N})$ steps as opposed to the classical requirement of $O(N)$ steps. The most celebrated example of quantum computing capabilities, however, is Shor's algorithm[2] which shows that a quantum computer can efficiently factor an arbitrarily large integer into primes in $O(\log^3(N))$ time.

In the classical computer, information is stored using bits - basic units of computation that can be in the state 0 or 1. Bits can be physically realized using an electrical switch or some other type of binary system. Clearly, when a bit is in the state 1, it cannot be in the state 0 and vice versa. This may

not seem like a very powerful system, but the capacity for information scales exponentially with the number of bits combined.

Quantum computing takes advantage of two-level quantum systems that can be in a superposition of a binary state in order to solve problems more efficiently than could be done classically. However, implementation of a quantum computer is difficult due to the fragile nature of quantum states, which are easily disrupted. It has been shown that quantum computing is theoretically possible through fault-tolerant quantum computation. Like classical machines, quantum computations will necessarily accumulate errors due to the inability to create perfect quantum gates - physical applications of unitary operations. Realistically, only a finite subset of the infinite number of gates available will be able to be implemented fault-tolerantly; that is, they can be implemented within an acceptably low margin of error. Therefore, the possibility of practical quantum computing relies on the ability to approximate all non-fault-tolerant gates using sequences of gates from the finite universal set. This Thesis deals with the problem of finding these fault-tolerant sequences.

The goal of this project is to create a program that simulates the application of the Solovay-Kitaev theorem - a very important result for the possible implementation of quantum computing. The program has useful applications as a teaching tool or learning aid for those interested in learning more about quantum computing. As a relatively new field, sources that cover material beyond the basics are difficult to find. Therefore, this program serves to fill a void in pedagogical resources on quantum computing.

Following this introduction, Section 2 provides a brief overview of the basics of quantum computing, including a description of quantum bits and quantum logic gates. A detailed proof of the Solovay-Kitaev theorem is provided on Section 3, followed by an introduction to topological quantum computing with Fibonacci anyons in Section 4. In Section 5, a description of the program is given with an explanation of possible applications and future work that is being planned.

2 Quantum Computing Basics

2.1 The Qubit

The usefulness of Quantum Computing as a field of study is two-fold. It is clearly worthwhile as a unique approach to studying and understanding quantum mechanical systems but it has also been shown that quantum algorithms can provide efficient solutions to some of the open problems in computer and information sciences. The most well-known example of this is certainly Shor's algorithm for integer factorization. Classically, many special-purpose algorithms and approximation schemes exist for this problem but an all-encompassing practical method has not been found. There are other notable quantum algorithms that indicate that quantum computing may not only provide solutions to difficult problems but significantly speed up operations that we can already perform within some tolerable approximation.

Quantum Computing offers a more powerful technique for computing by storing and manipulating data in quantum mechanical systems. The quantum analog of the classical bit is the qubit which can also be in the state 1 or 0. Unlike its classical counterpart, however, the qubit can instead occupy a superposition of 1 and 0. Physically, the state of a qubit may be represented by the spin of an electron, the charge state of a superconducting island, or any other system that allows for some sort of binary superposition. The state of a qubit is described by the wavefunction

$$|\psi\rangle = a|0\rangle + b|1\rangle \tag{1}$$

The values a and b are complex coefficients whose absolute values squared provide the probability of the qubit being in state 0 and 1 respectively. Furthermore, the values of a and b are restricted such that

$$|a|^2 + |b|^2 = 1 \tag{2}$$

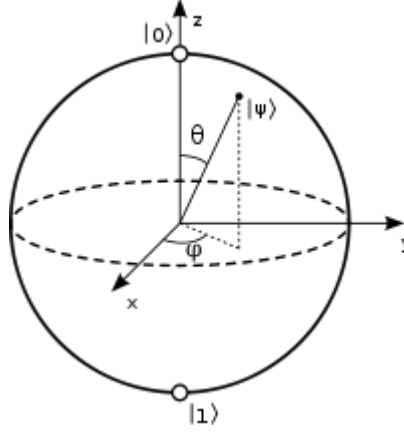


Figure 1: Bloch sphere representation of a qubit.

This normalization condition guarantees that the qubit will be either 0 or 1 upon measurement. The Bloch sphere (Fig.1) representation of a qubit is a visual representation that provides a good exposition of the difference between the classical bit and the qubit in terms of information-storing capacity. The north pole of the sphere represents a classical bit in the 0 state while the south pole represents the 1 state. These are the only points of the sphere within the reach of the classical bit; the qubit, on the other hand, has access to the entire sphere. Clearly then, the wavefunction of the qubit is expressible in terms of the angles θ and ϕ . Due to the normalization condition mentioned earlier, we can write the coefficients of the wavefunction as

$$a = e^{i\gamma} \cos \frac{\theta}{2} \quad (3)$$

$$b = e^{i\gamma} e^{i\phi} \sin \frac{\theta}{2} \quad (4)$$

We can remove the global phase factor $e^{i\gamma}$ to obtain a new wavefunction dependent on the parameters that produce a point on the Bloch sphere.

$$|\psi\rangle = \cos \frac{\theta}{2} |0\rangle + e^{i\phi} \sin \frac{\theta}{2} |1\rangle \quad (5)$$

These coefficients are representative of the spinor representing spin-up for the component of spin angular momentum along an arbitrary direction \hat{r} .

While one may think of the qubit as being in a superposition state, the only values that can be obtained upon measurement are 0 and 1. This is because observation of the qubit causes its wavefunction to collapse into one of its basis states. Clearly, one of the difficulties in quantum computing is devising a way to manipulate qubits whose states cannot be known.

2.2 Quantum Gates and Quantum Circuits

2.2.1 Quantum Gates

Similar to any classical computer, a quantum computer is also composed of circuitry and gates. Classically, circuitry in a computer is used to push information around while logic gates perform the most elementary computations on this information. The quantum analogs are the quantum circuit and the quantum gate. In this paper, we will be dealing exclusively with single qubit systems. This may seem prohibitively limiting but we will see that, as opposed to single bit classical systems, single qubit systems are rich in complexity and worthwhile to analyze on their own.

Classically, the only single bit logic gate is the NOT gate, which negates the input bit. That is, for an input of 1, it returns a 0 and vice versa. There is a quantum counterpart to this logic gate, which works by exchanging the probabilities amplitudes of the qubit. Let's say we have a qubit described by the following normalized wavefunction

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle \tag{6}$$

We can also represent this qubit by writing its coefficients in matrix form

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{7}$$

Inversion of the coefficients can be achieved by multiplying the matrix form of the wavefunction by the matrix

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad (8)$$

Therefore, we have

$$X|\psi\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix} \quad (9)$$

As opposed to the single classical option for a single bit gate, there are infinitely many single qubit gates. The common property of all of these gates is that they are represented by unitary matrices. That is, for each gate U , $U^\dagger U = I$ where I is the 2×2 identity matrix and the dagger indicates the Hermitian conjugate of the matrix. We can easily prove that all single-qubit gates must be unitary by observing that the wavefunction that represents the state of the qubit must remain normalized throughout the computation (that is, there must be a probability of 1 that it will be found in some position). So we begin with

$$\langle \psi | \psi \rangle = 1 \quad (10)$$

If we multiply the wavefunction by some matrix U then it must remain normalized so we can state

$$\langle U\psi | U\psi \rangle = \langle \psi | U^\dagger U \psi \rangle = \langle \psi | U^\dagger U | \psi \rangle = 1 \quad (11)$$

Because we know the original wavefunction is normalized, we can see clearly that UU^\dagger must be equal to the identity matrix. The requirement that the gates be unitary is, in fact, the only condition required of a single qubit gate and therefore, there are infinitely many gates that may be applied to a single qubit. Three notable single qubit gates are the Hadamard (H), phase (S),

and $\frac{\pi}{8}$ (T) gates.

$$H = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad (12)$$

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \quad (13)$$

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{pmatrix} \quad (14)$$

Additionally, the well-known Pauli matrices are viable single qubit quantum gates.

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad (15)$$

Of particular interest for us are the unitary operations that are defined in the following way

$$R_x(\gamma) = e^{-i\gamma\frac{\sigma_x}{2}} = \cos\left(\frac{\gamma}{2}\right)I - i\sin\left(\frac{\gamma}{2}\right)\sigma_x \quad (16)$$

$$R_y(\gamma) = e^{-i\gamma\frac{\sigma_y}{2}} = \cos\left(\frac{\gamma}{2}\right)I - i\sin\left(\frac{\gamma}{2}\right)\sigma_y \quad (17)$$

$$R_z(\gamma) = e^{-i\gamma\frac{\sigma_z}{2}} = \cos\left(\frac{\gamma}{2}\right)I - i\sin\left(\frac{\gamma}{2}\right)\sigma_z \quad (18)$$

These unitary operations represent rotations γ of the Bloch sphere vector about the x , y , and z axes respectively [3]. From these equations, we can obtain an equation for a rotation γ about an arbitrary axis in the \hat{n} -direction where $\hat{n} = (n_x, n_y, n_z)$.

$$R_{\hat{n}}(\gamma) = e^{-i\frac{\gamma}{2}\hat{n}\cdot\vec{\sigma}} = \cos\left(\frac{\gamma}{2}\right)I - i\sin\left(\frac{\gamma}{2}\right)\hat{n} \cdot \vec{\sigma} \quad (19)$$

If we say $a = \cos(\frac{\gamma}{2})$ and $\vec{b} = \sin(\frac{\gamma}{2})\hat{n}$ then we have

$$R_{\hat{n}}(\gamma) = aI - i\vec{b} \cdot \vec{\sigma} \quad (20)$$

Since any unitary operation that can be performed on a single qubit can be expressed as a rotation, we can see here that these matrices take the form of the equation above. Furthermore, because unitary matrices are subject to the constraint that UU^\dagger must be equal to the identity, we know that a and \vec{b} obey the following relation

$$a^2 + \vec{b}^2 = 1 \quad (21)$$

Therefore, any unitary operation acting on a single qubit may be expressed as a single vector \vec{b} .

2.2.2 Quantum Circuits

For a single qubit, the specification of the application of one or more unitary matrices constitutes a circuit. A single quantum NOT gate is represented in the following way

$$\text{---}\boxed{X}\text{---} \quad (22)$$

The application of the NOT gate on a single qubit is represented by the following quantum circuit diagram

$$|\psi\rangle \text{---}\boxed{X}\text{---} X|\psi\rangle \quad (23)$$

$$a|0\rangle + b|1\rangle \text{---}\boxed{X}\text{---} b|0\rangle + a|1\rangle \quad (24)$$

Similarly, the following representations are used for the Hadamard, phase, and $\frac{\pi}{8}$ gates

$$\text{---}\boxed{H}\text{---} \quad (25)$$

$$\text{---}\boxed{S}\text{---} \quad (26)$$

$$\text{---}\boxed{T}\text{---} \quad (27)$$

The following quantum circuits represent the application of these gates on single qubits

$$a|0\rangle + b|1\rangle \text{---}\boxed{H}\text{---} a\frac{|0\rangle + |1\rangle}{\sqrt{2}} + b\frac{|0\rangle - |1\rangle}{\sqrt{2}} \quad (28)$$

$$a|0\rangle + b|1\rangle \text{---}\boxed{S}\text{---} a|0\rangle + ib|1\rangle \quad (29)$$

$$a|0\rangle + b|1\rangle \text{---}\boxed{T}\text{---} a|0\rangle + be^{\frac{i\pi}{4}}|1\rangle \quad (30)$$

2.2.3 Two-Qubit Gates and Universal Gate Sets

In order to actually compute with a quantum computer it is not enough to simply act on each qubit separately with single qubit operations. It is also necessary to interact the qubits with each other in controlled ways in order to produce entangled states.

Remarkably, it can be shown that it is not necessary to implement a 2^d -dimensional matrix to perform operations on d qubits. We can decompose any 2^d -dimensional unitary matrix U into the form

$$U = U_0 U_1 \dots U_n \quad (31)$$

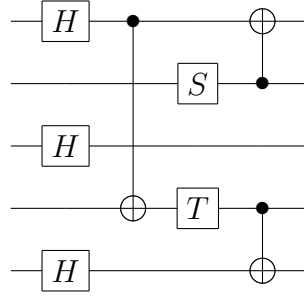
where U_k is a two-level unitary matrix. That is, any operation we want to perform on any number of qubits greater than two can be written as the product of two qubit or single qubit gates. Furthermore, it can be shown that with the help of the two-qubit controlled not (cNOT) gate it is possible to implement any arbitrary two-level unitary gate. Therefore, it is possible to implement any 2^d -level unitary matrix with a product of only single-qubit gates and the cNOT gate.

The cNOT gate works on two qubits by specifying a control qubit (in

this case, the top qubit) and a target qubit. If the control qubit is set to $|1\rangle$ then the NOT operation is performed on the target qubit; otherwise, nothing happens.


(32)

The set of all possible single qubit operations together with the cNOT gate form what is known as a universal set of gates. This means that all possible quantum algorithms can be expressed as a quantum circuit consisting of just these gates. A typical quantum circuit is shown below.


(33)

This thesis deals with a fundamental issue which arises in any practical scheme for realizing a universal set of gates for a quantum computer. In order to carry out quantum computation, it is necessary to minimize error propagation and decoherence due to effects from the environment using fault-tolerant quantum computation, a full discussion of which can be found in [4]. However, all such fault-tolerant schemes provide only a finite number of single qubit gates that can be implemented with the accuracy needed for reliable computation. A frequently mentioned set of universal gates is the Hadamard, Phase, $\frac{\pi}{8}$, and cNOT gates. But since we know that a truly universal gate set must include all single qubit gates, we must be able to show that we can approximate the infinite number of single qubit gates using the small finite set.

For any fault-tolerant quantum computer it will therefore be necessary to approximate a given desired single-qubit rotation as a sequence of single-

qubit gates drawn from the particular universal set that is available. An example of such a sequence is shown below

$$|\psi\rangle \text{---} \boxed{H} \text{---} \boxed{T} \text{---} \boxed{T} \text{---} \boxed{H} \text{---} \boxed{T^{-1}} \text{---} |\psi'\rangle \quad (34)$$

The sequence shown in this circuit diagram represents the operation

$$|\psi\rangle = T^{-1}HTTH|\psi'\rangle \quad (35)$$

where the product of these matrices yields the single matrix

$$T^{-1}HTTH = \begin{pmatrix} \frac{1+e^{\frac{i\pi}{2}}}{2} & \frac{1-e^{\frac{i\pi}{2}}}{2} \\ -i \sin \frac{\pi}{4} & \cos \frac{\pi}{4} \end{pmatrix} \quad (36)$$

While it is easy to find the resultant gate produced by a sequence of single-qubit operations, it is difficult to find the sequence that produces a specific gate. The simplest way to do this would be to perform a brute force search over all of the possible sequences from the finite set, but since the number of sequences grows exponentially with the length, the time required to find the sequence that models the desired gate accurately is likely to take longer than the lifetime of the universe.

It should be clear now that to implement a fault-tolerant quantum computer, it must be possible to not only approximate any single-qubit gate from a finite set of gates, it must be done so efficiently. Because *all* fault-tolerant quantum computing schemes provide a finite gate set, this is a problem that is universal for quantum computing, independent of the scheme that is used. Fortunately, there is an elegant solution to this problem. A powerful theorem known as the Solovay-Kitaev theorem, which is reviewed in the next section, provides an efficient method to find these sequences.

3 The Solovay-Kitaev Theorem

3.1 The Theorem

Because any unitary matrix can be considered a gate in quantum computing, it may seem like there are an innumerable many choices of gates for a quantum algorithm to pull from; however, this is not the case. Quantum circuits are limited to a smaller *fault-tolerant* subset of all the unitary matrices. This does not inhibit the power of the quantum machine, though. The Solovay-Kitaev theorem tells us that we can construct fault-tolerant approximations to these non-fault-tolerant gates using the smaller set. Formally, the Solovay-Kitaev theorem offers a guarantee that any gate that can be applied can be efficiently approximated to within some error ϵ_0 using a series composed of gates from the universal gate set. The length of the sequence varies with the given ϵ_0 as $\Theta(\log^c(\frac{1}{\epsilon_0}))$ [5].

The Solovay-Kitaev theorem is based on an algorithm which we refer to as the Solovay-Kitaev algorithm (and occasionally, simply as Solovay-Kitaev). Solovay-Kitaev finds an approximation U_1 to U in the following way. First a brute force search is performed over gate sequences of some predetermined length. For example, the program described in this paper which implements this algorithm, stores sequences of length nine for Fibonacci anyon braids (described in Sec. 4). Once the best brute force approximation, U_0 , is found, the matrix Δ is calculated where Δ is given by

$$\Delta = UU_0^{-1} = VWV^\dagger W^\dagger \quad (37)$$

The sequence $VWV^\dagger W^\dagger$ is the so-called group commutator of some V and W . The essential idea behind the Solovay-Kitaev algorithm is that by approximating the matrices V and W in the group commutator, one can find an approximation to Δ that is significantly better than what one would find by simply approximating Δ directly. By applying Solovay-Kitaev to the ma-

trices chosen for V and W , we can find the 9-length sequence approximations V_0 and W_0 to V and W . Then, we can construct a new sequence U_1 such that

$$U_1 = V_0 W_0 V_0^\dagger W_0^\dagger U_0 \quad (38)$$

This sequence, now of length 45, better approximates our desired operation U .

What was just described is referred to as “depth 1” Solovay-Kitaev. (“Depth 0” Solovay-Kitaev refers to the original brute force search over length 9 sequences). By recursively applying the Solovay-Kitaev algorithm, we can achieve better and better approximations to the original desired operation. At depth 2, we simply repeat the steps described above, but now replace the depth 0 approximations to U , V , and W with depth 1 approximations. More generally, at depth k we replace these approximations with depth $k - 1$ approximations. One can readily see that every time the depth increases by 1 the length of the sequence grows by a factor of 5, so the result of depth k Solovay-Kitaev is a sequence of length $l_0 5^k$ where l_0 is the length of sequences at depth 0 (as stated before, $l_0 = 9$ in our case). Remarkably, although the braid length grows exponentially with depth, the accuracy of the approximation improves *super-exponentially* with depth, leading to the logarithmic scaling described above. Below we outline a proof of this fact. Our discussion closely follows that which can be found in [3].

3.2 The Details

To understand Solovay-Kitaev, the properties of a universal gate set must be discussed. We’ll restrict the discussion to gates that operate on single qubits, represented by two dimensional unitary matrices. A simple example of a universal gate set would then be the Hadamard gate, H , and the $\pi/8$ gate, T . Let us then define G_l to be the set of all gate sequences of length up to l (for example, for the $\{H, T\}$ gate set a member of G_9 might be the

length 9 sequence $THTTHT^{-1}HTT$).

At this point, it is necessary to define $SU(2)$ which is the set of 2×2 unitary matrices with determinant 1. It should be noted that while the unitary matrices H and T used above are not in $SU(2)$, we can always multiply a unitary matrix operator by an arbitrary phase factor to bring it into $SU(2)$. Now, let's say our universal gate set G is a small subset of $SU(2)$. We say that G is dense in $SU(2)$ if any point in $SU(2)$ (here, a "point" is a 2×2 $SU(2)$ matrix) is a limit point of G_l or is in G_l . That is, points in $SU(2)$ either belong to G_l or they are arbitrarily close to a point in G_l such that any ϵ -neighborhood of a point in G_l contains a point in $SU(2)$ other than itself. Another way we can state this is to say that G_l forms an ϵ -net for $SU(2)$. Our interest then is in how large the sequence length l which defines G_l must become as ϵ is decreased.

To begin the proof of the Solovay-Kitaev theorem, it is necessary to establish a definition for distance between unitary operators. We will use the following as our measure for distance

$$D(A, B) = \text{tr}|A - B| \quad (39)$$

where $|X| = \sqrt{X^\dagger X}$. If we use our previous definition of a unitary operation, $U = aI - i\vec{b} \cdot \vec{\sigma}$, then can represent our unitary matrices as

$$U = \begin{pmatrix} a + ib_z & b_y + ib_x \\ ib_x - b_y & a - ib_z \end{pmatrix} \quad (40)$$

Therefore, we can express the distance D between U and U_0 as

$$D(U, U_0) = 2\sqrt{(a - a_0)^2 + (b_x - b_{x_0})^2 + (b_y - b_{y_0})^2 + (b_z - b_{z_0})^2} \quad (41)$$

Additionally, we will be making use of the group commutator

$$[AB]_{gp} = ABA^\dagger B^\dagger \quad (42)$$

Now, let S_ϵ be the set of all points S in $SU(2)$ with $D(S, I) \leq \epsilon$, where I is the identity matrix. Assume G_l is an ϵ^2 -net for S_ϵ . If we choose some x such that

$$S_x = e^{\frac{-i\vec{x} \cdot \vec{\sigma}}{2}} \quad (43)$$

Up to a small approximation, $||\vec{x}|| \leq \epsilon^2$ so we can choose \vec{y} and \vec{z} , each no more than length ϵ , such that $\vec{x} = \vec{y} \times \vec{z}$. We will also choose a \vec{y}_0 and \vec{z}_0 such that

$$U_{y_0} = e^{\frac{-iy_0 \cdot \vec{\sigma}}{2}} \quad (44)$$

$$U_{z_0} = e^{\frac{-iz_0 \cdot \vec{\sigma}}{2}} \quad (45)$$

are elements of G_l that approximate S_y, S_z to within ϵ^2 . We can write this concisely as

$$D(U_{y_0}, S_y) \leq \epsilon^2 \quad (46)$$

$$D(U_{z_0}, S_z) \leq \epsilon^2 \quad (47)$$

It then follows that the group commutator of U_{y_0} and U_{z_0} is given by

$$[U_{y_0}, U_{z_0}]_{gp} = e^{\frac{-iy_0 \cdot \vec{\sigma}}{2}} e^{\frac{-iz_0 \cdot \vec{\sigma}}{2}} e^{\frac{iy_0 \cdot \vec{\sigma}}{2}} e^{\frac{iz_0 \cdot \vec{\sigma}}{2}} \quad (48)$$

$$\simeq (1 - iy_0 \cdot \vec{\sigma}/2 + O(\epsilon^2))(1 - iz_0 \cdot \vec{\sigma}/2 + O(\epsilon^2)) \\ (1 + iy_0 \cdot \vec{\sigma}/2 + O(\epsilon^2))(1 + iz_0 \cdot \vec{\sigma}/2 + O(\epsilon^2)) \quad (49)$$

$$\simeq (1 - i\vec{y}_0 \times \vec{z}_0 \cdot \vec{\sigma}/2 + O(\epsilon^3)) \quad (50)$$

which is within a distance ϵ^3 of U_{x_0} . To derive this, we use the properties of the Pauli matrices (in particular $(\vec{a} \cdot \vec{\sigma})(\vec{b} \cdot \vec{\sigma}) = (\vec{a} \cdot \vec{b})I + i(\vec{a} \times \vec{b}) \cdot \vec{\sigma}$).

Using the triangle inequality, we can put an upper bound on $D(S_x, [U_{y_0}, U_{z_0}]_{gp})$ as

$$D(S_x, [U_{y_0}, U_{z_0}]_{gp}) \leq D(S_x, U_{x_0}) + D(U_{x_0}, [U_{y_0}, U_{z_0}]_{gp}) \quad (51)$$

$$\simeq |\vec{x} - \vec{x}_0| + d\epsilon^3 \quad (52)$$

$$\simeq |\vec{y} \times \vec{z} - \vec{y}_0 \times \vec{z}_0| + d\epsilon^3 \quad (53)$$

$$\leq C\epsilon^3 \quad (54)$$

where C is a constant.

Now, if we let $S_{\sqrt{(C\epsilon^3)}}$ be the set of all points S in $SU(2)$ with $D(S, I) \leq \sqrt{(C\epsilon^3)}$, then G_l is an ϵ^2 -net for $S_{\sqrt{(C\epsilon^3)}}$. For some U_0 in G_l , we have $D(S, U_0) \leq \epsilon^2$ and SU_0^\dagger is an element of S_{ϵ^2} . Then we can find some W and V in G_l so that we have $D([W, V]_{gp}, SU_0^\dagger) \leq C\epsilon^3$. The key result is

$$D([W, V]_{gp} U_0, S) \leq C\epsilon^3 \quad (55)$$

$$D(WVW^\dagger V^\dagger U_0, S) \leq C\epsilon^3 \quad (56)$$

In other words, we can construct a new $U_1 = WVW^\dagger V^\dagger U_0$ that is an element of G_{5l} and is a $C\epsilon^3$ -net for $S_{\sqrt{(C\epsilon^3)}}$. This process can be applied iteratively to create better approximations to S . After k applications of the approximation, we obtain an element of $G_{5^k l}$ that is an $\epsilon(k)^2$ -net for $S_{\epsilon(k)}$ where $\epsilon(k)$ is given by

$$\epsilon(k) = \frac{(C\epsilon)^{\frac{3^k}{2}}}{C} \quad (57)$$

If the original ϵ is such that $C\epsilon < 1$, then we see that $\epsilon(k)$ will get small rather quickly as we increase k . Now, if we want to approximate some sequence to within ϵ_0 (that is, we enter ϵ_0 into the program), we need to choose k , the number of times the algorithm iterates, such that $\epsilon(k)^2 < \epsilon_0$. Using the definition of $\epsilon(k)$, this relation gives us the following result

$$\frac{3^k}{2} < \frac{\log(\frac{1}{C^2\epsilon_0})}{2\log(\frac{1}{C\epsilon})} \quad (58)$$

Given that we know $l(k) = 5^k l$, we can use the preceding relation to find that the length of the approximation should be

$$l(k) = \frac{3^{kc}}{2} l < \left(\frac{\log(\frac{1}{C^2 \epsilon_0})}{2 \log(\frac{1}{C \epsilon})} \right)^c l \quad (59)$$

where $c = \frac{\log(5)}{\log(\frac{3}{2})}$. This completes the proof of the Solovay-Kitaev theorem by showing that the length of the gates required to approximate any unitary matrix to within some ϵ_0 is $\Theta(\log^c(\frac{1}{\epsilon_0}))$. The key result of the Solovay-Kitaev theorem tells us that even though the increase in length of the gate sequence is exponential, the improvement in $\epsilon(k)$ is super-exponential [3]. Therefore, the polylogarithmic increase in length is acceptable for the purposes of approximating a non-fault-tolerant gate. More details on the Solovay-Kitaev theorem can be found in [5].

4 Fibonacci Anyons

4.1 Braiding

Due to the fact that there are many different kinds of systems and particles that are subject to the laws of quantum mechanics, there are many different models that propose a method for implementing a quantum computer. In this paper, special interest will be paid to topological quantum computing using Fibonacci anyons. The topological quantum computing method presented here is based on [6] and [7], in which a more detailed analysis can be found.

Non-abelian anyons are defined as particles that live in two-dimensions whose wavefunction evolution is dependent on how they are braided around one another. The braiding process can be represented as the multiplication of unitary matrices. Fibonacci anyons are a subset of this group with a spin-like attribute that takes on the values 1 and 0, which are clearly useful as basis states for quantum computing. The process by which Fibonacci q-spin is combined is known as fusion; fusion rules dictate that combinations of Fibonacci anyons may take on a total q-spin of 1 or 0 or a superposition of the two. For computational purposes, qubits are encoded using 3 Fibonacci

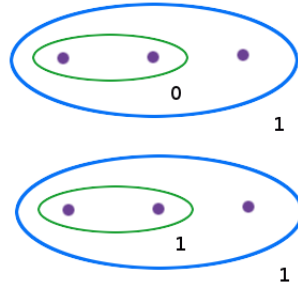


Figure 2: Computationally acceptable states for 3 Fibonacci anyons. An inner q-spin of 0 corresponds to a basis state of $|0\rangle$ while an inner q-spin of 1 corresponds to a basis state of $|1\rangle$.

anyons with a total q-spin of 1, where the state of the qubit is determined by the total q-spin of the two left-most (or bottom-most) anyons. The computationally acceptable states are shown in Fig. 2. A state where the total q-spin of the three anyons is 0 is not computationally acceptable.

For a single qubit represented using three Fibonacci Anyons, there are 4 ways of braiding the quasi-particles around on another. The 4 elementary braids are shown in Fig. 3.

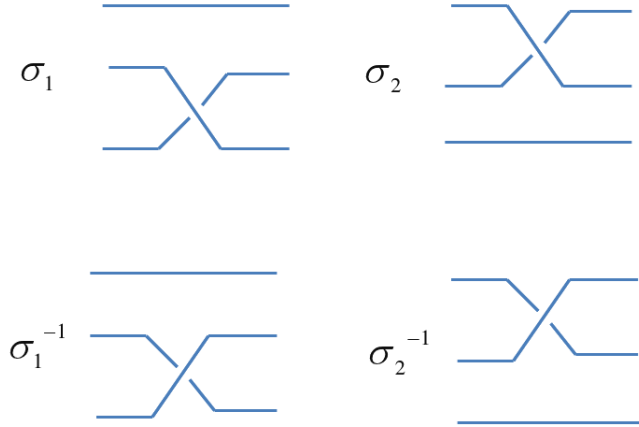


Figure 3: Elementary braid operations.

Each individual braid is equivalent to multiplying the state of anyons by a unitary matrix, so one can easily see that each way of braiding the anyons around one another constitutes a quantum gate. From this, we derive a universal gate set for the case of computing with Fibonacci anyons with the braids identified as σ_2 , σ_1 , σ_2^{-1} and σ_1^{-1} [8], where the matrix representations are given as

$$\sigma_1 = \begin{pmatrix} e^{-\frac{7\pi i}{10}} & 0 \\ 0 & e^{\frac{7\pi i}{10}} \end{pmatrix} \quad (60)$$

$$\sigma_2 = \begin{pmatrix} -\tau e^{-\frac{i\pi}{10}} & \sqrt{\tau} e^{-\frac{i\pi}{10}} \\ \sqrt{\tau} e^{-\frac{i\pi}{10}} & -\tau e^{\frac{i\pi}{10}} \end{pmatrix} \quad (61)$$

To approximate a unitary operation that is not represented in the 4 elementary braids, braids are constructed using the universal set of braids. This is equivalent to applying the unitary operations in a sequence. An example of a small braid is shown in the figure below.

Given a simple braid, it is easy to compute the matrix operation that corresponds to the braid in its entirety; we only need to multiply the matrices of the elementary braids of which the longer braid is composed. However, the problem in topological quantum computing is that we actually need to find the braid that corresponds to a given matrix. This is where we can apply Solovay-Kitaev using our four elementary braids to approximate any unitary operation.

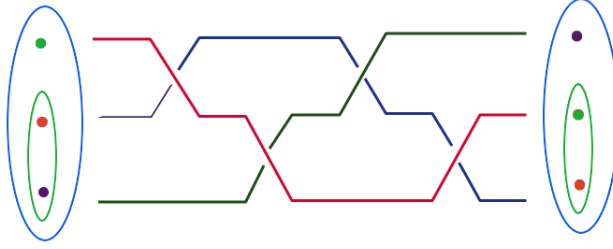


Figure 4: Demonstration of a simple braid. In order, the elementary braids represented are σ_2 , σ_1 , σ_2^{-1} and σ_1^{-1} .

4.2 Weaving

More efficient searches for the best approximation of a unitary operation may be performed by searching over a smaller search space composed of a subclass of braids known as weaves. Weaves are braids in which only one of the quasi-particles (known as the 'weft' quasi-particle) is woven around the other two stationary 'warp' quasi-particles. The benefit of search over weaves is that each sequence is topologically unique, whereas all combinations

of braid sequences of a certain length give rise to multiple appearances of the same unitary operation [8]. The elementary weave operations are shown in the figure below.

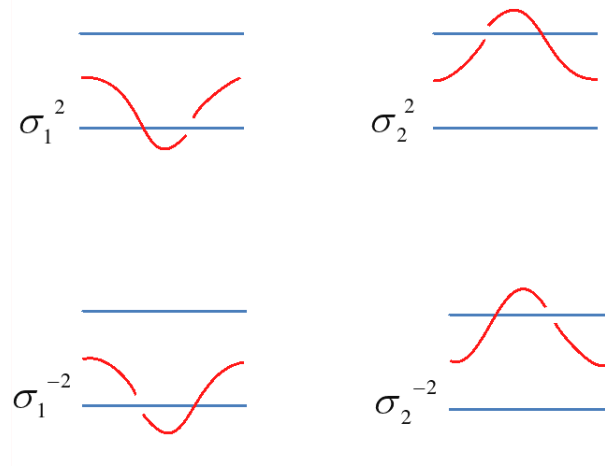


Figure 5: Elementary weave operations.

5 The Program

5.1 Introduction

The program presented is designed to provide an environment for testing and observing the application of the Solovay-Kitaev algorithm on different quantum computing paradigms. Particular interest is paid to topological quantum computing as the program also includes an environment for observing the effect of the Solovay-Kitaev theorem on braids and weaves produced by Fibonacci anyons. The program itself is interactive, allowing the user to manipulate key variables in the environment, and intuitive. The two primary features of the program are the ability to apply the Solovay-Kitaev theorem to any desired depth using various universal gate sets and the ability to create visual representations of the Solovay-Kitaev algorithm on topological paradigms.

The program is written in Python 2.7.2 and was developed on a Ubuntu 11.04 machine. Additional tools used include NumPy 1.5.1, SciPy 0.9.0, Matplotlib 1.0.1, sqlite3 0.7.7 and PyQt4. When run, the program renders a graphical user interface that allows the user to enter information about a unitary matrix that they wish to approximate using the specified universal gate set.

5.2 Environment

The program is designed to be intuitive and easy to use for those that have some understanding of the basics of quantum computing and the goals of the Solovay-Kitaev algorithm. The first screen encountered by the user is shown in Fig. 6. The interface enables the user to choose between the tabbed options of “Fibonacci Anyon Rotations” or “Other Universal Rotations”. The latter allows the user to choose from a predefined selection of universal gate sets with which to perform the Solovay-Kitaev algorithm. The user enters

information about the unitary operation they would like to approximate and the results of the computation are displayed on the right-hand side. The Fibonacci Anyon Rotation tab is selected by default and displays the same functionality with the added feature of observing the braid or weave that results.

The screenshot shows a web application titled "Quantum Gate Compiler". It has two tabs: "Fibonacci Anyon Rotations" (selected) and "Other Universal Rotations".

Under the "Fibonacci Anyon Rotations" tab, there is a text area with instructions: "Please enter information about the unitary gate you would like to approximate. You can either enter a 2-dimensional unitary matrix or a b-vector. Note that the time required to perform the calculation will increase exponentially with depth." Below this are four bullet points: "• Square roots can be entered as `math.sqrt()`", "• Fractions can be expressed using `/`", "• Complex i can be expressed as `'i'`", and "• To express e, use `'math.exp(x)'`".

There are two radio buttons: "Braids" (selected) and "Weaves".

Below the radio buttons, it says "Enter the elements of a 2-dimensional unitary matrix:". There are four input fields arranged in a 2x2 grid. Below these is a "Depth:" label followed by an input field. A "Submit" button is to the right.

Below the "Submit" button, it says "Enter a b-vector that characterizes a unitary operation:". There are three input fields labeled b_z , b_{z^2} , and b_{z^3} . Below these is a "Depth:" label followed by an input field. A "Submit" button is to the right.

On the right side of the interface, there are three large empty rectangular boxes. Below these, it says "Matrix result:" and "Distance from desired unitary matrix:". A mouse cursor is visible near the bottom right of the interface.

Figure 6: First input screen.

5.2.1 Fibonacci Anyon Rotations

The default tab selection of 'Fibonacci Anyon Rotations' allows the user to see how the Solovay-Kitaev theorem can determine the braids and weaves of anyon worldlines that could be used to implement quantum computing. Fibonacci Anyons were chosen as a featured scheme in this program due to the promise that topological quantum computing shows as an implementable paradigm for quantum computing. However, the gate model of quantum computing encompasses many different schemes; remarkably, the Solovay-Kitaev

theorem is valid for all of them.

The initial screen contains an input section of the left-hand side and an initially blank output section of the right-hand side. The user must first choose whether they want to display a braid pattern or a weave pattern. The user is then given the option of entering a unitary matrix or vector that describes the unitary matrix. In both cases, a depth is entered as well. A depth value of 0 corresponds to a single brute force search for the closest sequence of nine gates. A depth value of 1 corresponds to a single application of the Solovay-Kitaev algorithm. Essentially, the depth is the number of times that the Solovay-Kitaev algorithm is applied recursively to the sequence. With the increase in precision comes an increase in time required to perform the computation. The user is therefore notified that at greater depths, the program may take quite a long time to run. Alternative search methods to speed up the computation are discussed in the Applications and Future Work section of this document.

Once the user has entered data, clicking the submit button will begin the computation with the input type that the button is associated with. In both cases, data is checked for validity before proceeding with the computation and the user is notified if the input is not syntactically correct or if it does not constitute a unitary matrix. The computation is then performed, ranging in time from a few seconds for a depth of 0 to several hours for a depth of 6 and beyond.

The output of the computation (Fig. 7) is displayed on the right in the three empty boxes. The first box displays a visual representation of the resulting braid or weave in the style of Fig. 3 of this document. The second box displays the sequence of rotations in symbolic form. To accommodate longer sequences a scroll bar appears in both boxes, allowing the user to view the entire sequence. The last box displays the resultant matrix and the ϵ -value that corresponds to the inaccuracy of the approximation.

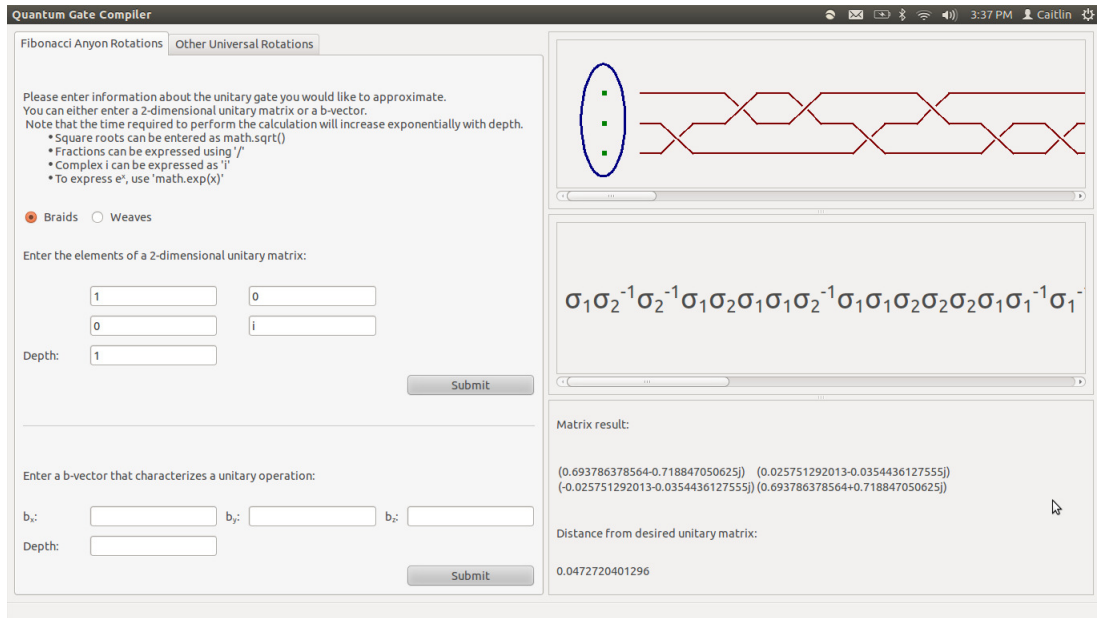


Figure 7: The output screen after selecting Fibonacci Anyon Braids and entering the Hadamard gate to be approximated.

Once a computation has been performed and the results displayed, input data is preserved in the form to allow the user to increase the depth size and rerun the computation quickly or make slight changes to previously entered data.

5.2.2 Other Universal Rotations

The alternate tab selection, “Other Universal Rotations” allows the user to see how the Solovay-Kitaev theorem may be applied to any universal gate set in the gate model of quantum computation. As seen in Fig. 8, the input section differs from the Fibonacci anyon tab only in that the user is supplied a drop-down list of pre-implemented universal gate sets to choose from. Once a set is chosen, the input is entered as described in the previous section. The results section is identical as well except for the top box, which is left blank. Discussed in the Future Work section is the ability for users to specify a new universal gate set that they can implement and add to the drop-down list.

The screenshot shows a web application titled "Quantum Gate Compiler". It has two tabs: "Fibonacci Anyon Rotations" and "Other Universal Rotations", with the latter being selected. The interface is divided into two main columns. The left column contains input fields and instructions. The right column contains two large empty boxes for results.

Left Column:

- Instructions: "Please enter information about the unitary gate you would like to approximate. You can either enter a 2-dimensional unitary matrix or a b-vector. Note that the time required to perform the computation will increase exponentially with depth." It also lists supported syntax: "Square roots can be entered as `math.sqrt()`", "Fractions can be expressed using `/`", "Complex i can be expressed as `'i'`", and "To express e, use `'math.exp(x)'`".
- Gate Set Selection: A dropdown menu labeled "H, S, and T".
- Matrix Input: "Enter the elements of a 2-dimensional unitary matrix:" followed by four input fields arranged in a 2x2 grid.
- Depth Input: "Depth:" followed by an input field.
- Submit Button: A grey button labeled "Submit".
- B-vector Input: "Enter a b-vector that characterizes a unitary operation:" followed by three input fields labeled b_1 , b_2 , and b_3 .
- Depth Input: "Depth:" followed by an input field.
- Submit Button: A grey button labeled "Submit".

Right Column:

- Top Box: A large empty rectangular box.
- Bottom Box: Labeled "Matrix result:" and "Distance from desired unitary matrix:". It contains a large empty rectangular box and a mouse cursor icon in the bottom right corner.

Figure 8: Second input screen.

5.3 Results

In this section, the results of some typical calculations using the Fibonacci anyons elementary braids as the universal gate set are recorded. These results serve to prove how accurate the prediction of Solovay-Kitaev is.

The Hadamard gate we will be using here is represented by the following matrix:

$$\begin{pmatrix} \frac{i}{\sqrt{2}} & \frac{i}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} & \frac{-i}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} .707i & .707i \\ .707i & -.707i \end{pmatrix} \quad (62)$$

Ideally, as we approximate this gate at greater depths, the elements of the approximate matrix should move closer to the values of the elements of the matrix above. For the Hadamard gate, at depth 0 (brute force search through length 9 braids), the approximating braid and corresponding matrix are

$$\sigma_2 \sigma_1 \sigma_2 \sigma_1^{-1} \sigma_2^{-1} \sigma_1^{-1} \sigma_2^{-1} \sigma_1^{-1} \sigma_2^{-1}$$

$$\begin{pmatrix} .618i & .786i \\ .786i & .618i \end{pmatrix} \quad (63)$$

To improve this approximation, a search needs to be made over longer sequences. The search, however, quickly becomes infeasible as the sequence gets longer. Instead, the Solovay-Kitaev theorem provides a smarter way of searching. At depth 1 (one application of Solovay-Kitaev), the resultant braid and matrix are

$$\sigma_1 \sigma_2^{-1} \sigma_2^{-1} \sigma_1 \sigma_2 \sigma_1 \sigma_1 \sigma_2^{-1} \sigma_1 \sigma_2 \sigma_1 \sigma_2 \sigma_1^{-1} \sigma_1^{-1} \sigma_2^{-1} \sigma_1^{-1} \sigma_2^{-1} \sigma_2^{-1} \sigma_1^{-1} \sigma_2 \sigma_1^{-1} \sigma_1^{-1} \sigma_2^{-1} \sigma_1^{-1}$$

$$\sigma_2 \sigma_2 \sigma_1^{-1} \sigma_2 \sigma_2 \sigma_1 \sigma_2 \sigma_1 \sigma_1 \sigma_2^{-1} \sigma_1^{-1} \sigma_2^{-1} \sigma_2 \sigma_1 \sigma_2 \sigma_1^{-1} \sigma_2^{-1} \sigma_1^{-1} \sigma_2^{-1} \sigma_1^{-1} \sigma_2^{-1}$$

$$\begin{pmatrix} .086 + .736i & .067i \\ .671i & .086 - .736i \end{pmatrix} \quad (64)$$

The sequence we achieve at depth 1 is a length 45 sequence. An important point to be made here is the a brute force search over braids of length 45 would take an *extremely* long time to perform. These results show how, using the Solovay-Kitaev theorem, we can improve approximations as we need to in order to compute fault-tolerantly and do so efficiently.

In the first table, the distance between the desired unitary operation and the approximate sequence is shown for depths up to five.

Depth	Hadamard Gate	NOT Gate	Phase Gate
0	.199088	.112766	.140251
1	.097284	.110181	.047272
2	.033484	.070979	.013649
3	.009710	.029425	.033024
4	.005581	.005909	.008034
5	.000683	.000532	.000896

Table 1: Table of distances between desired gates - Hadamard, NOT, and Phase - and the approximation made at the indicated depth.

In the second table, we focus on the Hadamard gate to observe another method of determining accuracy. We know that any matrix M has the property $MM^{-1} = I$, where I is the identity matrix. Therefore, if we want to determine how accurately some matrix M approximates another matrix N , we can calculate NM^{-1} and see how close it comes to the identity matrix. At depth k , we take the inverse of the approximation U_k made to the Hadamard gate, H , and we get $\Delta_{Hadamard} = HU_k^{-1}$ which should more closely resemble the identity matrix as the depth increases.

Depth	Hadamard Gate Δ	
0	$\begin{bmatrix} .992909 & .118877 \\ -.118877 & .992909 \end{bmatrix}$	
1	$\begin{bmatrix} .995268 + .060639i & -.045689 + .060639i \\ .045689 + .060639i & .995268 - .060639i \end{bmatrix}$	
2	$\begin{bmatrix} .999439 + .012370i & -.028272 + .012983i \\ .028272 + .012983i & .999439 - .012370i \end{bmatrix}$	
3	$\begin{bmatrix} .999953 - .003084i & .001472 - .009089i \\ -.001472 - .009089i & .999953 + .003084i \end{bmatrix}$	
4	$\begin{bmatrix} .999984 + .000904i & -.002802 - .004741i \\ .002802 - .004741i & .999984 - .000904i \end{bmatrix}$	
5	$\begin{bmatrix} .999999 + .000332i & -.000209 - .000558i \\ .000209 - .000558i & .999999 - .000332i \end{bmatrix}$	

Table 2: Table of $\Delta_{Hadamard}$ matrices for the depth indicated.

5.4 Applications and Future Work

The primary application of the program is that of a teaching/explanatory tool. As a relatively new field, pedagogical resources on quantum computing for students are few in number, especially regarding more advanced topics. This program serves as a contribution that may be used for self-teaching or as an aid for instructors.

More peripheral applications include using the program as an aid to studying how searches for an approximation may be made more efficiently or more effectively. Future work on this project is to include development of more sophisticated searching algorithms. For example, the database that stores N -length gate sequences for Fibonacci anyons scales in size as 4^N . The simplest way to search for the best approximation among the entries is to perform a brute-force search, which can be done at best in linear time. For large databases, this can be unacceptably slow, especially as the algorithm is run at greater depths. Minimizing the time required to make a search involves minimizing the search space as much as possible. To do this, one must be familiar with the physical representation of the distance between states.

A fairly easy way to solve this problem is to apply a sophisticated Closest-Pair algorithm. The algorithm works by breaking up the searchable space into subspaces with characteristic points and then comparing the desired value with the characteristic points. The characteristic point that is closest to the value represents the first subspace to search. The closest point in this subspace to the desired value is stored. However, it is possible that a point from an adjacent subspace that lies close to the boundary of the first subspace may, in fact, be the closest point. From here, two solutions may be considered. If the distance between the desired point and the closest found point from the first subspace is d , then the easier solution involves searching all adjacent subspaces for a point that is within a distance $\leq d$ of the desired point. A more efficient solution involves searching only the space within the

distance d of the first subspace. If the searchable space is broken up into enough subspaces, the speedup in searching should be significant.

As mentioned in the previous section, future work on this project is to include the ability for users to specify and implement additional universal gate sets. Currently, the program is packaged with pre-built databases that contain all combinations of a universal gate set up to a certain length. An additional interface is to be built into the program which allows the creation and management of databases, built by the user, which can be used in the program. In this way, the program is extensible and can be used to perform the Solovay-Kitaev theorem on any universal gate set the user desires. This feature may be useful in the future for comparing the effectiveness of the algorithm with respect to the set that is used.

6 References

- [1] L.K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. Proceedings, STOC 1996. arXiv:quant-ph/9605043v3.
- [2] P.W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. Proceedings of the 35th Annual Symposium on Foundations of Computer Science (1994). arXiv:quant-ph/9508027v2.
- [3] I.L. Chuang and M.A. Nielsen. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.
- [4] J. Preskill. Fault-Tolerant Quantum Computation. *Introduction to Quantum Computation* (1997). arXiv:quant-ph/9712048v1.
- [5] C.M. Dawson, M.A. Nielsen. The Solovay-Kitaev algorithm. arXiv:quant-ph/0505030.
- [6] M. Freedman, M. Larson, Z. Wang. A Modular Functor which is Universal for Quantum Computation. Communications in Mathematical Physics, 227:605622, 2002. arXiv:quant-ph/0001108.
- [7] A.Yu. Kitaev. Fault-tolerant Quantum Computation by Anyons. Annals of Physics 303, 2 (2003). arXiv:quant-ph/9707021.
- [8] N.E. Bonesteel, L. Hormozi, G. Zikos, and S.H. Simon. Braid Topologies for Quantum Computation. Phys. Rev. Lett. 93, 140501 (2004). arXiv:quant-ph/0505065.
- [9] L. Hormozi, G. Zikos, N.E. Bonesteel, and S.H. Simon. Topological Quantum Computing. Phys. Rev. B 75, 165310 (2007). arXiv:quant-ph/0610111.
- [10] N.D. Mermin. *Quantum Computer Science: An Introduction*. Cambridge University Press, 2007.