Joren W. Brunekreef

Topological Quantum Computation and Quantum Compilation

Supervisors:
Dr. Joost Slingerland
Prof. dr. Cristiane de Morais Smith

Universiteit Utrecht

NUI Maynooth
Ollscoil na hÉireann, Má Nuad

Abstract

A central problem in realizing universal quantum computers is dealing with decoherence effects, which can destroy the vulnerable quantum states present in the internal registers. Several schemes have been developed that provide resistance to decoherence, leading to theories of fault-tolerant quantum computation. One such scheme is the theory of topological quantum computation, which encodes qubit states in robust topological properties of certain quantum systems. However, fault-tolerant theories typically only provide a finite set of primitive gates, from which we must then form approximations to arbitrary quantum gates. The Solovay-Kitaev algorithm describes a procedure for accomplishing this, and proves that the number of primitive gates required to approximate an arbitrary gate is only polylogarithmic in the desired degree of accuracy. In this thesis, we first outline the basic theory of quantum computation. Secondly, we describe the general problem of quantum compilation, list several possible strategies for relevant optimizations, and outline a proof of the Solovay-Kitaev theorem. Next, we give an introduction to the theory of topological quantum computation by anyons, with special focus on Fibonacci-anyons. Finally, we list some numerical results of an implementation of the Solovay-Kitaev algorithm when applied to a Fibonacci-anyon model and analyze the performance gains obtained through applications of the proposed optimizations.

Contents

Introduction

Very interesting theory - it makes no sense at all.
- Groucho Marx

Many great minds have, at some point, struggled to come to terms with the counter-intuitive predictions made by quantum theory. Its laws simply seem so far off from what we experience in our daily lives, that it might initially seem absurd to replace the clean and intuitive picture of classical mechanics by its unwieldy quantum counterpart. Our everyday experiences suggest that objects are always in a definite state - for example, we are either home or at work, either dead or alive, and either sitting or standing. In the realms of quantum mechanics, however, this need not be the case: on tiny scales, objects can be in multiple states at once. That is, up to the point that we actually try to determine which of these states it is in - upon measurement, its so-called *superposition* will disappear and we find only one of the possible 'classical' states. Even though this might sound more like magic than science, it turns out that quantum mechanics is by far the most accurate theory in all of contemporary physics. It is nearly impossible to grasp its intricacies on an intuitive level, but the mathematical framework that physicists have built for it over the years is rigorous and firm.

Over the years, physicists started wondering whether it might be possible to tap into this 'hidden' world and harness some of the powers that drive it. This led to the creation of the theory of *quantum computation*, where the classical ideas of computation are extended to make use of quantum physics. It turns out that this new paradigm seems to be significantly more powerful than the classical models of computation that we are used to. For example, Grover's quantum search algorithm allows one to perform a full search over a hundred drawers, by only inspecting ten. However, while ordinary computers are fairly easy to construct, building a physical realization of a quantum computer brings a whole host of new challenges. The most demanding of these is the problem of *decoherence*, which causes quantum superpositions to be lost through nearly unavoidable interactions with the environment. A remedy to this has been proposed in the form of *topological quantum computation*, in which we stretch our imagination even further and perform quantum computing operations by essentially braiding the histories of quasi-particles in a two-dimensional environment. Such a system would effectively do away with almost all of the decoherence-related difficulties, with one downside - the physical implementation of a topological quantum computer still seems years away.

This thesis aims to discuss the contemporary understanding of topological quantum computers, and the additional mathematical machinery needed to implement arbitrary quantum algorithms on such a machine. As stated before, intuition is already of little use from early on, so the focus is mainly on the mathematical constructs on which the theory is based. It has even been said that the highest level of understanding quantum mechanics is knowing how to use the mathematical machinery, and not worrying about the seemingly unnatural conclusions you obtain along the way.

The paradox is only a conflict between reality and your feeling what reality ought to be.
- Richard Feynman

Chapter 1

Quantum computation

1.1 Qubits

We first present a basic introduction to the theory of quantum computation. A more complete and detailed discussion can be found in [1] or [2].

In classical computation, the fundamental unit of information is called a *bit*. A bit can take on two distinct *states* - usually denoted 0 and 1. In quantum computation, the equivalent of a bit is called a *qubit*. The states corresponding to 0 and 1 could be denoted $|0\rangle$ and $|1\rangle$ in 'qubit terminology' and are examples of so-called *basis states* - orthogonal quantum states that can be mapped to some observable.

However, these two states are merely two of the possible options in the infinite amount of available qubit states: qubits, obeying the laws of quantum mechanics, can take on any linear combination of states. We generally call such a linear combination a *superposition*. In Dirac notation, we write the state of such a qubit in superposition as

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1.1}$$

Here α and β are the entries of a unit vector in a two-dimensional complex vector space (which is also a two-dimensional Hilbert space). More simply put, α and β are complex numbers satisfying $|\alpha|^2 + |\beta|^2 = 1$. This so-called *normalization constraint* has its roots in basic quantum mechanics, and the reason for it will soon become apparent.

So far, it seems that such a qubit is infinitely more potent than a classical bit: there is, after all, an infinite amount of complex numbers α, β that satisfy the normalization constraint. Naively, one would say that it is now possible to encode an arbitrary amount of information in the decimal expansion of one of these two parameters. However, this is where quantum mechanics stops cooperating. When one *measures* a qubit (in the $|0\rangle, |1\rangle$-basis), the result is merely 0 or 1 - the quantum mechanical wave function *collapses* and leaves the qubit in one of two states $|0\rangle, |1\rangle$.

This is the point where the parameters α and β come in: when a qubit in the state (1.1) is measured, the result is 0 with probability $|\alpha|^2$ and 1 with probability $|\beta|^2$. This is also the motivation for the normalization constraint: the sum of all probabilities must equal 1. We call the complex numbers α, β the *probability amplitudes* associated with a specific state.

1.1.1 Visual representation of a single qubit state

In order to comfortably visualize the state of a qubit, we first make a few observations. Firstly, all qubit states are unit vectors in a two-dimensional complex vector space, which means we can 'portray' it in a four-dimensional Euclidean space. Secondly, we point out that we can multiply a qubit state by a so-called *global phase factor* $e^{i\gamma}$ ($\gamma \in \mathbb{R}$) without changing its observable properties - after all, $\left|e^{i\gamma}\alpha\right|^2 = |\alpha|^2$, so the probability of measuring a qubit in one of its two basis states is unchanged. Thirdly, the normalization constraint allows us to write

$\alpha = \cos\frac{\theta}{2}, \beta = e^{i\varphi}\sin\frac{\theta}{2}$ (this expression may seem arbitrary at first, but it will soon make perfect sense).

Putting all this together, we arrive at a new expression for the state of a qubit:

$$|\psi\rangle = e^{i\gamma}\left(\cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle\right) \tag{1.2}$$

This expression is more complicated on first sight, but it actually allows us to picture the state in three dimensions. Since the global phase factor is irrelevant for our purposes, we can 'peel off' one dimension of the state space and view the state as a unit vector in three-dimensional Euclidean space, which is a point on the familiar 2-sphere. The angles θ and φ serve to describe the exact position of the point on the sphere. This (partial) visualization of the qubit state space is often called the *Bloch sphere*, after physicist Felix Bloch.

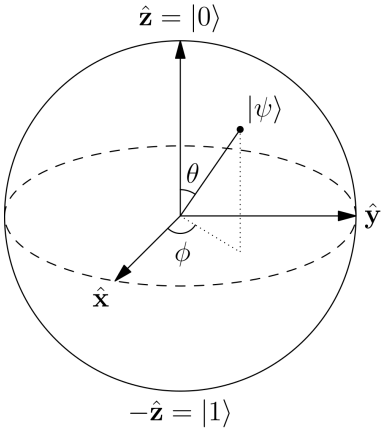


Figure 1.1: Bloch sphere visualization of a single qubit state.

1.1.2 Multiple qubits

When one has access to multiple bits, more information can be encoded. Since the possible values of a bit form the space $\{0, 1\}$, the space of possible values of an n-bit system is equal to the Cartesian product $\{0, 1\}^n$, which has cardinality 2^n. For example, a 2-bit system could be in one of the states 00, 01, 10 or 11.

For qubits, the situation is different. A 2-qubit system would have $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$ as possible basis states. The general state of a 2-qubit system can therefore be written in the form

$$|\varphi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \tag{1.3}$$

Again, the four probability amplitudes must satisfy the normalization constraint: $|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$.

Now suppose we start out with two 'separate' qubits in the states $|\varphi\rangle = (\alpha_0|0\rangle + \beta_0|1\rangle)$ and $|\psi\rangle = (\alpha_1|0\rangle + \beta_1|1\rangle)$ respectively. We can now write their aggregate state as follows:

$$|\varphi\rangle \otimes |\psi\rangle = \alpha_0\alpha_1|00\rangle + \alpha_0\beta_1|01\rangle + \beta_0\alpha_1|10\rangle + \beta_0\beta_1|11\rangle$$

We call the state $|\varphi\rangle \otimes |\psi\rangle$ the *tensor product state* of the states $|\varphi\rangle$ and $|\psi\rangle$. Obviously, the state of an n-qubit system is therefore described by a vector in the tensor product space of the single qubit two-dimensional complex Hilbert spaces. This tensor product space is again a complex Hilbert space, now of dimension 2^n.

While one can always describe the joint state of multiple qubits in this tensor product space, the converse is generally false: most unit vectors in the 2^n dimensional complex Hilbert

space cannot be decomposed into n two-dimensional unit vectors with respect to the tensor product. Take, for example, the state

$$|\varphi^+\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right)$$

It is simple to show that this state cannot be written as the tensor product of two single qubits. We call this type of state an *entangled* state. Entanglement is a key element of the theory of quantum information - an important example of its applications is the phenomenon of quantum teleportation [3].

1.2 Qubit operations

Operations on classical bits are performed by applying *logic gates*, which take a collection of bits as input and produce a (possibly different) amount of output bits. In quantum computation, we employ a similar notion of logic gate, which we call a *quantum gate*. However, we have to make slight changes to the original framework: while classical logic gates are fully specified by their so-called *truth table*, this does not hold for their quantum counterpart.

Take, for example, the classical `NOT` gate. Its truth table is as follows:

input	output
0	1
1	0

Table 1.1: Truth table for a classical `NOT` gate

However, in order to apply this gate to a qubit, we have to take into account that such a qubit can also be in a superposition of $|0\rangle$ and $|1\rangle$. A good candidate for the quantum `NOT` gate would be a gate that 'exchanges' the probability amplitudes, so that it would send the state (1.1) to $\beta|0\rangle + \alpha|1\rangle$. It turns out that this is indeed a consistent extension of the classical `NOT` operation to the world of quantum information.

1.2.1 Quantum gate matrices

In the previous sections, we have described that the state of a qubit (or an n-qubit system) is described by a unit vector in a 2^n-dimensional complex Hilbert space. Since we can linearly transform vectors through multiplying them by matrices, it is reasonable to describe operations on qubits by matrices that preserve the norm of the vectors they act on. In the circuit model of quantum computation, these matrices are known as *quantum gates*. It turns out that the exact family of gates we need for expressing operations on n-qubit states is the set of $2^n \times 2^n$ *unitary* matrices, which are precisely the matrices U for which $U^\dagger U = I$, with $U^\dagger$ being the Hermitian adjoint (= conjugate transpose) of U.

Here we see another hint of the richness of quantum computation: while there are only four possible operations on a single classical bit (the identity, the `NOT` gate, and the two irreversible operations $x \to 0$ and $x \to 1$), we now have a continuum of unitary matrices that all constitute a valid single-qubit operation.

Three of the most important quantum gates are the *Pauli gates*, defined as

$$X \equiv \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y \equiv \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z \equiv \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{1.4}$$

Upon closer inspection, we see that the Pauli-X gate is precisely the quantum version of the `NOT` gate we described earlier. Another frequently used gate can be obtained by adding the X and Z gates together (and renormalizing to obtain a unitary matrix), producing the *Hadamard* gate:

$$H \equiv \frac{X+Z}{\sqrt{2}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{1.5}$$

If we look at the action of the Hadamard gate on the $|0\rangle$ and $|1\rangle$ basis states, we see that it produces a state 'halfway' between $|0\rangle$ and $|1\rangle$. It is easily checked that this is again an orthonormal basis, which we call the $|+\rangle, |-\rangle$ (or Hadamard) basis.

$$\begin{aligned} H|0\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \qquad =: |+\rangle \\ H|1\rangle &= \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \qquad =: |-\rangle \end{aligned} \tag{1.6}$$

By observing that all Pauli matrices are self-inverse ($A^2 = 1$), we can exponentiate them to obtain the following expression:

$$e^{iAx} = \cos(x)I + i\sin(x)A \tag{1.7}$$

Now if we denote the vector of Pauli matrices (X, Y, Z) by $\vec{\sigma}$ and a real unit vector $\hat{n} = (n_x, n_y, n_z)$, we can define a matrix

$$R_{\hat{n}}(\theta) \equiv e^{-\frac{i\theta}{2}\hat{n}\cdot\vec{\sigma}} = \cos(\frac{\theta}{2})I - i\sin(\frac{\theta}{2})\hat{n}\cdot\vec{\sigma} \tag{1.8}$$

This matrix $R_{\hat{n}}(\theta)$ defines an arbitrary rotation of a qubit state vector on the Bloch sphere by an angle θ around the axis $\hat{n}$. It turns out that *all* possible single-qubit gates can be expressed in this form. By filling in π as the angle (and the appropriate basis vector $\hat{n}$), we retrieve the standard Pauli matrices (up to an unimportant global phase).

It will soon become convenient to graphically represent consecutive qubit operations. For this, we can use *circuit diagrams*. In these diagrams, qubits 'live' on imaginary horizontal wires. Gates can then operate on one or more of these wires. Note that this approach is very similar to the classical circuit model of computation. There are, however, a few important distinctions. Firstly, quantum circuits cannot exhibit loops. Secondly, classical circuits have gates that can 'merge' two or more wires into one, which is illegal in quantum circuits (it violates unitarity). The last restriction is less obvious, but it is also illegal to 'copy' a qubit (`FANOUT` in classical terms). This is due to a slightly more involved law called the *no-cloning theorem* [4], which essentially proves that it is impossible to duplicate arbitrary quantum states.

As an example, Figure 1.2 shows a qubit to which the X and H gates are applied consecutively. The complete operation performed on a qubit by a sequence of gates can also be represented by the product of all the associated gate matrices. Note that this multiplication has to be applied in reverse, as the rightmost matrix in a matrix product acts on the qubit first.

$$\alpha|0\rangle + \beta|1\rangle \text{ ─}\boxed{X}\text{─}\boxed{H}\text{─ } \alpha\tfrac{|0\rangle-|1\rangle}{\sqrt{2}} + \beta\tfrac{|0\rangle+|1\rangle}{\sqrt{2}}$$

$$HX = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$$

Figure 1.2: An example quantum circuit and its associated matrix operation.

1.2.2 Multi-qubit operations

An essential ingredient for computing in general is the ability to do *controlled operations*. A classical example of a controlled operation is flipping one input bit if the other input bit is

1 - often called the CNOT. Just as before, it is possible to define quantum analogues of these controlled operations. Obviously, controlled operations necessarily work with multiple input qubits, so the previously used 2×2-matrices will not suffice anymore.

The matrix for a general controlled-U gate (assuming that the first qubit is the control qubit) is obtained by taking the direct sum ($\oplus$) of the 2×2 identity matrix and the U gate. For example, the CNOT gate is found as follows:

$$I \oplus X = \left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array}\right) \tag{1.9}$$

The CNOT gate can be used to construct *entangling* operations. After applying such an operation to certain two-qubit input states, the output state will not be separable w.r.t. the tensor product. This intuitively makes sense: if the control qubit was in a strict superposition of its basis states, we could say that the NOT operation was only 'partially' applied to the target qubit. As a result, the target's state is now inextricably linked to the control state. The circuit symbol for the CNOT gate is as follows:

Figure 1.3: Circuit symbol for CNOT gate

For completeness, we also describe how to form the matrices that constitute 'independent' operations on multiple qubits. Take, for example, a two-qubit system, in which we want to leave the first qubit alone but apply a Hadamard operation on the second. This amounts to applying the identity gate I on the first qubit and H on the second. The matrix that represents this operation can be found by computing the tensor product $I \otimes H$, which is a 4×4 matrix that acts on the tensor product state space of the two input qubits.

With these tools at hand, we are able to produce a very interesting and useful two-qubit circuit (shown in Figure 1.4): the one that produces the maximally entangled *Bell states*. The EPR-pairs in the 1935 paper by Einstein, Podolsky, and Rosen [5] are examples of 'qubits' in a Bell state - the authors originally attempted to use them to prove that quantum mechanics was incomplete (if locality was to be preserved). They argued that either locality would be violated by some nonlocal interaction (which is highly undesirable), or that there must be some 'hidden' information that quantum mechanics did not describe, thereby making it an incomplete theory.

Ironically, John S. Bell (after whom the states are named) subsequently used these states in his 1964 paper, "On the Einstein Podolsky Rosen paradox" [6], to prove one of the most profound results in quantum mechanics: the Bell inequality. It effectively shows that any conceivable 'local hidden variable' theory is inconsistent with quantum mechanics. The theorem essentially establishes a sharp divide between the classical and quantum world, and even provides an experiment that can be used to decide which of these is the 'true' picture. So far, all experiments have produced results in favor of quantum mechanics.

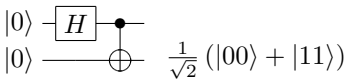


Figure 1.4: Circuit diagram for producing the Bell state $|\Phi^+\rangle$ from input state $|00\rangle$.

1.3 Motivation for quantum computation

So far, we have established how to formally describe and operate on an n-qubit system: the state of such a system is fully specified by 2^n complex numbers, representing the probability amplitudes of its basis states. Now suppose (optimistically!) that we were able to store one such amplitude classically in every atom in the universe. Current observations indicate that there are around 10^{80} atoms in the observable universe, so it would not even be possible to store the state of a mere 400-qubit quantum computer classically by any means!

However, we also stated that it is not directly possible to 'read out' these complex amplitudes: measuring a quantum system will only return a specific basis state, with outcome probabilities specified by these amplitudes - a process that also 'destroys' the quantum state. Furthermore, quantum processes are very susceptible to *decoherence* ('interaction with the environment'), a phenomenon that can introduce noise up to the point that the full quantum mechanical superposition is lost [7]. Some schemes have been developed that can deal with small amounts of decoherence, leading to *fault-tolerant* quantum computation. This can either be achieved on the 'circuit-level', or on the physical level. Chapter 3 will discuss a scheme that introduces robustness to errors on the physical level, called *topological quantum computation* [8].

Therefore, it seems that all initially apparent advantages are immediately lost due to this wave function collapse. The key observation to make here, is that only *measurement* (which can be seen as intentionally decohering a system) is destructive to a quantum state. As long as the state is not disturbed by such a process, its evolution remains fully quantum mechanical. In a way, one could say that in such an undisturbed 400-qubit system, the Universe continuously does calculations on these 2^{400} complex numbers, only to reveal part of the final answer upon measurement.

Even if we can only see a part of this 'hidden' process, it sounds very reasonable to ask whether it might be possible to tap into this seemingly astronomical computational power. It turns out that there seem to be very specific classes of problems that can be solved significantly 'faster' when using a quantum approach as opposed to a classical (or even a classical probabilistic) one. Even though a lot of evidence points towards the affirmative, it is still an open question whether this is, in fact, true. There is no doubt that some quantum algorithms perform better than the best known classical counterparts, but it has not yet been ruled out that these classical algorithms can still be improved significantly. An interesting example is *Shor's algorithm* [9], a procedure that can factor integers into its prime factors in polynomial time. The best known classical factoring algorithm works in sub-exponential time (about $O(e^{1.9(\log N)^{1/3}(\log\log N)^{2/3}})$), but there is as of yet no proof that a polynomial-time classical integer factorization algorithm cannot exist. In order to make this notion precise, we would have to make use of classical and quantum complexity theory, which is outside the scope of this thesis. A more detailed discussion can be found in [1].

Chapter 2

Quantum compilation

2.1 Universal sets of gates

Suppose we succeed in constructing an n-qubit register. If we now want to be able to perform any conceivable quantum algorithm on this register, we need to have access to all possible n-qubit gates. This amounts to the full set of $2^n \times 2^n$ unitary matrices ($U(2^n)$), up to a global phase factor. Any set of gates that can generate this complete set of operations is called *universal* with respect to quantum computation. A well-known (and significantly smaller) set of universal gates is the full set of single qubit operations ($U(2)$), together with the `CNOT` gate. Being able to perform these basic operations allows us to perform any quantum algorithm on a given qubit register, regardless of the actual number of qubits [1].

It turns out that we can reduce the number of necessary gates even further. For this, we first need to define the $\pi/8$-gate (usually denoted by T). Note that it actually includes a complex exponent of $\pi/4$, so the nomenclature is somewhat unfortunate, but we will keep to it for historical reasons[1].

$$T \equiv \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} \tag{2.1}$$

It has been proven that the Hadamard and $\pi/8$ gates can be used to approximate any single qubit gate to arbitrary accuracy [1]. Therefore, the set $\{H, T, \texttt{CNOT}\}$ is also universal. However, this set does not yet suffice for practical purposes.

2.2 Fault-tolerant quantum computation

Operations on quantum states are delicate processes and are often disrupted by noise. Therefore, schemes have been developed to perform *fault-tolerant* quantum computation. This can be achieved by encoding single qubits from the original algorithms in a block of multiple qubits, introducing a certain amount of redundancy to the computation. An example of such a scheme is the *Steane code* [10], which employs seven qubits for encoding the original single qubit state. The original gates to be performed on the single qubit must now also have a fault-tolerant counterpart that acts on the multiple-qubit block that encodes for the original. These gates must effectively suppress small errors that occur along the way and prevent them from propagating to other parts of the system. Typically, fault-tolerant schemes only provide a finitely generated set of possible operations [1].

Because of this, we need a reliable method of approximating arbitrary operations, based on such a finite set of available gates. As we have seen before, determining the full operation performed by a given finite circuit is easily accomplished by computing the matrix product of the applied gates. However, it is far more difficult to solve the reverse problem: "given a certain unitary matrix and a finite set of available gates, determine the circuit that performs

[1] Its origin can be seen by multiplying by the global phase $e^{-i\pi/8}$.

this operation to a given accuracy". Several methods are available for accomplishing this, but in order to continue, we first have to make the notion of 'approximation' more precise.

2.3 Error in quantum gate approximations

If our available universal set of gates is finite, it is obvious that we cannot implement any arbitrary gate exactly. The easiest way to see this, is that the set of all matrix products w.r.t. the input set is countably infinite, while the complete set of possible gates is a continuum and therefore uncountably infinite. This means that we can generally only approximate a desired gate up to arbitrary accuracy (if the set is, indeed, universal). In order to quantify this accuracy, we have to define a notion of distance between operators. There are several distance measures for matrices, but for our purposes the *trace distance* will suffice. The trace distance between two matrices U, V is defined as follows:

$$D(U,V) \equiv \operatorname{tr}|U-V| \tag{2.2}$$

$$|X| \equiv \sqrt{X^\dagger X} \tag{2.3}$$

Here the square root S of a matrix U is defined such that $S^2 = U$. Note that we will use a slightly different norm in the numerical calculations: there we compute the square root of the trace. Since all norms on finite dimensional vector spaces are equivalent, the two will differ by at most a constant factor, so that the same reasoning still applies.

Furthermore, we have previously mentioned that quantum states and gates are equivalent if they differ by a global phase $e^{i\gamma}$. Therefore, we can restrict our view from the full set of unitary matrices to the set of *special unitary* matrices ($SU(2^n)$), which is comprised of the unitary matrices with determinant 1. Every matrix in $U(2^n)$ can be multiplied by a global phase in order to obtain its $SU(2^n)$ counterpart [2].

For simplicity, we further restrict our view to single qubit gates - that is, matrices in $SU(2)$. It can be shown that every matrix $M \in SU(2)$ is of the form

$$M = \begin{pmatrix} \alpha & -\overline{\beta} \\ \beta & \overline{\alpha} \end{pmatrix} \quad \alpha,\beta \in \mathbb{C}, \quad |\alpha|^2 + |\beta|^2 = 1 \tag{2.4}$$

Now we must be careful to 'rotate' all the matrices we are dealing with into $SU(2)$, as omitting this could result in finding non-zero trace distances between matrices that are fully equivalent up to global phase. For example, the $SU(2)$ version (up to sign) of the Hadamard gate is

$$-iH = \frac{i}{\sqrt{2}} \begin{pmatrix} -1 & -1 \\ -1 & 1 \end{pmatrix}$$

We can now fully state the definition of the error in approximation:

Definition (Approximation error)**.** *The gate U is a δ-approximation for a target gate V if their representations in $SU(2)$ have trace distance $D(U,V) \leq \delta$.*

When only taking single-qubit gates and restricting ourselves to their representation in $SU(2)$, we arrive at a slightly more tangible condition for the set of input gates to be universal: they must generate a *dense set* in $SU(2)$, meaning that we can always find points in the generated set that are arbitrarily close to any desired matrix $M \in SU(2)$.

[2] This can be achieved by dividing the original matrix by the square root of its determinant.

2.4 Brute force method

The most straightforward method of finding approximations of arbitrary gates is a plain brute-force search. We start with a specific input set of gates (which should generate a dense set in $SU(2)$ if we want to guarantee success) and recursively walk the entire 'tree' of matrix products level by level. For example, if the base set is $\{H, T\}$, the first level of the walk is just the base set again. On the second level, the obtained products are $\{HH, HT, TT, TH\}$. After n recursive calls, this procedure has provided us with all possible matrix products involving at most n instances of the base matrices - we say that we have generated all matrices with words up to length n.

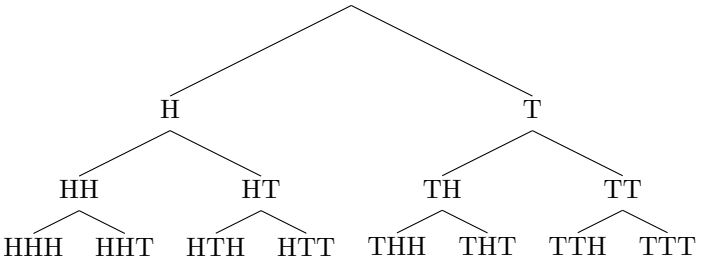


Figure 2.1: Example tree of matrix products up to depth 3.

If the input set generates a dense set in $SU(2)$, we can be sure that this search 'succeeds' at some point. For this, it needs two inputs: the target gate to be approximated and the maximum allowed error in the approximation. Now every time a new matrix product is calculated, we compute the trace distance between the generated gate and the target. If this distance is smaller than the allowed error, the procedure halts and returns the word associated with the approximation. This word could be seen as the 'compiled' version of the desired gate in terms of the available gates.

However, this strategy quickly becomes fairly infeasible if we require high degrees of accuracy. We can develop some intuition for this statement by reconsidering equation (2.4), which implies that $SU(2)$ forms a space that is diffeomorphic to the 3-sphere of radius 1. Therefore, the amount of 'patches' of radius δ will scale with δ^{-3}. If we want to be able to find a δ-approximation for an arbitrary gate, we must therefore have a search space of available matrix that has a size of (at the very least) $C\delta^{-3}$, for a certain constant C. Also, the base matrices will usually contain some 'trivial' relations - take, for example, the $\pi/8$ gate (T). The bottom right element is an 8th root of unity, implying that $T^8 = I$. Many 'branches' in the full tree will therefore exhibit duplicate behavior. Furthermore, it is very unlikely that an arbitrary base set (even if it generates a dense set) fills up $SU(2)$ uniformly, which complicates matters even more.

A possible partial remedy for this is including some heuristics. It is, for example, fairly simple to stop walking a certain branch if the outcome at that point is the identity matrix. This will allow the algorithm to search somewhat 'deeper' in the same amount of time, but in practice the gain is hardly spectacular. We could also try to record every computed matrix and stop walking a branch if the matrix at that node is already present in the 'history' collection. However, this brings another drawback: every computed matrix now has to be compared against the history set, which is computationally straining. In practice, this adjustment will even cause large slowdowns compared to the naive tree walk. We will see later that the latter 'comparative' method does have some advantages in specific situations, but it is not very useful for generic compilation.

2.5 The Solovay-Kitaev algorithm

In the previously mentioned brute-force method, the search for a target gate is completely *undirected* - we naively try all possible products we can form and hope for the best. A powerful theorem due to R. Solovay (unpublished) and (independently) A. Kitaev [11] provides a procedure that makes use of the geometric structure of $SU(2)$ in order to progressively enhance approximations to a target gate. The procedure - from now on the *Solovay-Kitaev algorithm* - takes as input a target gate, a desired accuracy and a so-called ϵ-net for $SU(2)$. An ϵ-net for a set V is a set U such that every point of V is within a distance ϵ of some point in U. This means that we first have to perform a pre-processing step - generating an ϵ-net from a finite set of available fault-tolerant gates.

A reasonable strategy is using the aforementioned approach of recursively walking the tree of matrix products, but now up to a feasible depth. The obtained matrices are stored along the way, together with their associated words. If the base set generates a dense set in $SU(2)$, the resulting collection will be an ϵ-net, unfortunately with unknown ϵ. It is far from trivial to determine the smallest possible ϵ so that the collection still is an ϵ-net for all of $SU(2)$. However, it turns out that the Solovay-Kitaev algorithm does not require very low upper bounds on ϵ: values around ~ 0.3 seem to suffice. It is fairly easy to check whether candidate nets meet this condition. One could, for example, compute a set of regular 'grid points' on the 3-sphere that form an $\epsilon/2$-net (after applying the natural homomorphism from the 3-sphere to $SU(2)$), and find $\epsilon/2$-approximations to these in the generated collection. If every grid point has such an approximation, the collection is surely an ϵ-net for $SU(2)$. Another strategy - less computationally intensive, but also less reliable - would be to randomly scatter points on the 3-sphere and apply the same 'pairing-up'. If the pairing is successful for a sufficiently large amount of points, it becomes increasingly likely that the collection is an ϵ-net.

Kitaev's proof of the theorem outlines a general strategy of progressively enhancing the accuracy of a matrix approximation, but the followed patterns do not fit well for real-world implementations. Instead, we describe an implementation of the algorithm (following the same ideas) proposed by Dawson and Nielsen [12]. Their article outlines a recursive algorithm, that calls itself to iteratively improve the approximation. This means we can execute it up to a given depth. Suppose we want to approximate a certain gate U. The 'depth-0' function simply performs a search over the input ϵ-net that we generated on beforehand, returning the closest match to the target gate it can find. We call this approximation U_0. We then compute the product $\Delta = UU_0^\dagger$. Since U and U_0 are unitary, this product will be a matrix that lies in the neighbourhood of the identity. We can then (non-uniquely) decompose this product in terms of two matrices V, W, such that

$$\Delta = UU_0^\dagger = VWV^\dagger W^\dagger \quad V, W \in SU(2) \tag{2.5}$$

The next step is finding approximations from our initial ϵ-net to these matrices V and W, calling them V_0 and W_0. We see that if we substitute these into equation (2.5) and multiply by U_0 on the right, we obtain $U \approx V_0 W_0 V_0^\dagger W_0^\dagger U_0$. The main result by Solovay and Kitaev is that this will be a better approximation to U than U_0. We then call this new approximation U_1. For the next approximation, we again compute the product $\Delta = UU_1^\dagger$ and decompose it in the same way as before. However, instead of finding direct approximations to V, W, we now apply a depth-1 Solovay-Kitaev algorithm to V and W themselves. With the resulting approximations to V and W, we form the word for the depth-2 approximation U_2 in the same way as before. It should now be clear how to perform the algorithm up to arbitrary depth.

An important drawback of the procedure is now easy to spot: with every iteration, the word length grows by a factor of 5. Therefore, it is very likely that the algorithm does not return the *shortest* possible δ-approximation to a desired gate (in terms of word length). After all, large parts of the product tree are simply 'skipped over'. We also see that the word length increases exponentially in the amount of iterations. However, a key conclusion of the Solovay-Kitaev theorems is that the accuracy of the approximation will increase super-exponentially, making it a very useful strategy for generic compilation.

2.5.1 Mathematical justification

Before we state the key theorem behind the Solovay-Kitaev algorithm, we define G_l to be the set of all products (w.r.t. a base set G) with word length $\leq l$.

Theorem (**Solovay-Kitaev**). *Let G be a finite set of elements of $SU(2)$ containing its own inverses, such that G generates a dense set in $SU(2)$. Let $\epsilon > 0$ be given. Now G_l is an ϵ-net in $SU(2)$ for $l = O(\log^c(\frac{1}{\epsilon}))$, with $c \approx 4$.*

Note that the theorem in itself provides no method of approximation - it merely states that the net G_l tightens quickly (meaning that G_l is an ϵ-net for exponentially decreasing ϵ) with polynomially growing word length l. The proof of the theorem allows for a general approximation procedure to be extracted. We now discuss the key points in the theorem (following the reasoning in [1]).

The first ingredient to the proof is a lemma that roughly states that G_l tightens especially quickly around the identity matrix. We first define S_ϵ to be the subset of $SU(2)$ such that $\forall U \in S_\epsilon : D(U, I) \leq \epsilon$.

Lemma. *Let G be a finite set of elements of $SU(2)$ containing its own inverses, such that G generates a dense set in $SU(2)$. There exists an ϵ_0 (independent of G), such that for any $\epsilon \leq \epsilon_0$ for which G_l is an ϵ^2-net for S_ϵ, the set G_{5l} is an $O(\epsilon^3)$-net for $S_{O(\epsilon^{3/2})}$.*

So, suppose we have generated all matrices with words up to length l and this turns out to be an ϵ^2-net for the set of all matrices within a distance $\epsilon \leq \epsilon_0$ of the identity, then the set G_{5l} containing all matrices with words up to five times longer will always be a tighter (namely, $O(\epsilon^3)$) net for a somewhat smaller (namely, $O(\epsilon^{3/2})$) neighbourhood around the identity.

The proof of this lemma hinges on a useful property of the so-called *group commutator* of two matrices $V, W \in SU(2)$

$$[V, W]_{\text{gp}} \equiv VWV^\dagger W^\dagger \tag{2.6}$$

We recall from equation (1.8) that we can write all special unitary matrices as $\exp(-i\vec{a} \cdot \vec{\sigma}/2)$ for some real vector $\vec{a}$ (note that we have absorbed the factor θ, so $\vec{a}$ is not necessarily of unit length). Now suppose that V, W are both close to the identity and are described by vectors $\vec{a}, \vec{b}$. This implies that $\vec{a} \cdot \vec{\sigma}$ has trace close to zero (look at the Taylor expansion). Now choose ϵ such that $\text{tr}\,|\vec{a} \cdot \vec{\sigma}|, \text{tr}\,|\vec{b} \cdot \vec{\sigma}| \leq \epsilon$. By writing out the Taylor expansions of $[V, W]_{\text{gp}} = VWV^\dagger W^\dagger$ and $\exp(-\left[\vec{a} \cdot \vec{\sigma}, \vec{b} \cdot \vec{\sigma}\right])$ (with $[A, B]$ being the regular matrix commutator $AB - BA$), we derive that

$$D\left([V, W]_{\text{gp}}, \exp(-\left[\vec{a} \cdot \vec{\sigma}, \vec{b} \cdot \vec{\sigma}\right])\right) = O(\epsilon^3) \tag{2.7}$$

From this, we see that the group commutator is fairly well approximated by the regular matrix commutator in the neighbourhood of the identity. Now we employ a useful identity relating to the Pauli matrices, which states that

$$\left[\vec{a} \cdot \vec{\sigma}, \vec{b} \cdot \vec{\sigma}\right] = 2i\left(\vec{a} \times \vec{b}\right) \cdot \vec{\sigma} \tag{2.8}$$

Substituting this into equation (2.7), we conclude that

$$D\left([V, W]_{\text{gp}}, \exp(-i(\vec{a} \times \vec{b}) \cdot \vec{\sigma}/2)\right) = O(\epsilon^3) \tag{2.9}$$

Now choose an arbitrary matrix $U \in S_{\epsilon^2}$. We can then find a vector $\vec{x}$ such that $U = \exp(-i\vec{x} \cdot \vec{\sigma})$. Since $SU(2)$ is locally Euclidean (it is a manifold), we can approximate the trace distance to the identity by the normal Euclidean distance $\|\vec{x}\|$ provided that $D(U, I)$ is small. Now since U is a matrix within a distance ϵ^2 from the identity, we can state that $\|\vec{x}\| \leq \epsilon^2$ to a good approximation. We can then write $\vec{x}$ as a product $\vec{y} \times \vec{z}$, for vectors $\vec{y}, \vec{z}$

(these are not unique, since we have a certain rotational freedom in choosing them) having norm at most ϵ. Since G_l is an ϵ^2-net for S_ϵ, we can find ϵ^2-approximations $\vec{y_0}, \vec{z_0} \in G_l$ to $\vec{y}, \vec{z}$.

Plugging the group commutator $[\exp(-i\vec{y}\cdot\vec{\sigma}/2), \exp(-i\vec{z}\cdot\vec{\sigma}/2)]_{\text{gp}}$ into equation (2.7), we see (after some algebra) that it is an $O(\epsilon^3)$-approximation to U. Since this applies for any matrix $U \in S_{\epsilon^2}$, we see that the set of all these commutators forms an $O(\epsilon^3)$-net for S_{ϵ^2}.

Now given an arbitrary matrix $V \in S_{C_1\sqrt{\epsilon^3}}$, we can find an ϵ^2-approximation $V_0 \in G_l$ (recall that G_l is an ϵ^2-net for S_ϵ and therefore surely so for $S_{C_2\epsilon^{3/2}}$). This, in turn, implies that $VV_0^\dagger \in S_{\epsilon^2}$. Now by the same reasoning as before, it is possible to find matrices $W_1, W_2 \in G_l$ such that $D([W_1, W_2]_{\text{gc}}, VV_0^\dagger) = O(\epsilon^3)$. Multiplying both factors in the distance function on the right by V_0 implies that

$$D([W_1, W_2]_{\text{gc}} V_0, V) = O(\epsilon^3) \tag{2.10}$$

This completes the proof of Lemma 2.5.1. The final step in proving Theorem 2.5.1 is very similar to the last step in the proof of the Lemma: we first find an $\epsilon(0)^2$-approximation $U_0 \in G_l$ to any matrix $U \in SU(2)$. Now compute $\Delta = UU_0^\dagger$. By the properties of the trace distance, we can derive that $D(\Delta, I) < \epsilon(0)^2$. Now set $\epsilon(1) = \epsilon(0)^2$. By applying Lemma 2.5.1, we see that we can find a $\Delta_1 \in G_{5l}$ which is an $\epsilon(1)^2$-approximation to Δ. From the definition of Δ, we see that ΔU_0 is then an $\epsilon(1)^2$-approximation to U, now of maximal length $l + 5l$. Iteratively applying this procedure gives progressively better approximations for U.

We note that the maximum word length after k iterations is therefore $l + 5l + \cdots + 5^k l$, which is always smaller than $\frac{5}{4}5^k l$. Now suppose we want to achieve some arbitrary accuracy ϵ. We must then find k such that

$$\epsilon(k)^2 < \epsilon \tag{2.11}$$

Through the iterated application of Lemma 2.5.1, we see that $\epsilon(k) = O(\epsilon(0)^{(3/2)^k})$. We can rewrite this asymptotic relation as follows:

$$\epsilon(k) = \frac{(C\epsilon(0))^{(3/2)^k}}{C} \tag{2.12}$$

for some constant C. We now plug equation (2.12) into (2.11) to obtain (after some algebra)

$$\left(\frac{3}{2}\right)^k < \frac{\log(1/(C^2\epsilon))}{2\log(1/(C\epsilon(0)))} \tag{2.13}$$

We can now combine this relation and the inequality for l to find that the maximum number of required gates is smaller than

$$\frac{5}{4}\left(\frac{\log(1/(C^2\epsilon))}{2\log(1/(C\epsilon(0)))}\right)^c l = O(\log^c 1/\epsilon), \quad c = \frac{\log 5}{\log(3/2)} \approx 4 \tag{2.14}$$

This completes the proof of the theorem.

2.6 Optimizations

2.6.1 GNAT-search

One expensive operation that has to be performed many times in the Solovay-Kitaev algorithm is a *search* for the best approximation to a certain gate in the available ϵ-net. Since we are looking for the best possible approximation, we have to compare the search target to every single gate in this collection if we use a naive, linear search. This can quickly become a bottleneck if the input ϵ-net contains many elements. Therefore, we could significantly speed up the algorithm by implementing the search method more efficiently.

An important fact to note at this point is that the search space is extremely *structured*. All elements are members of the group $SU(2)$, which is diffeomorphic to the 3-sphere. Intuitively,

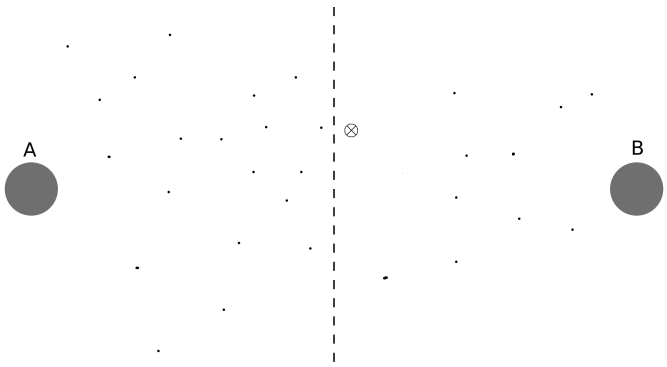


Figure 2.2: Example of a situation in which the GNAT-search will return a suboptimal result. The target gate to be searched for is denoted $\otimes$.

we could say that we only have to search for the best possible approximation to a target gate in some area 'around' it. To accomplish this, we use an approach loosely based on the methods proposed in [13].

This method is called the *geometric nearest-neighbour access tree search*, or GNAT-search. It works by first converting the unstructured ϵ-net into an access tree data structure and consequently using this tree structure in the search procedure. If done correctly, this will drastically reduce the number of necessary matrix comparisons.

The procedure for generating the access tree is recursive in nature. It takes as input an unstructured collection of special unitary matrices. From this collection, it randomly[3] selects a specified amount of n matrices that serve as nodes. Each of the remaining matrices is then compared to these nodes and consequently assigned to the node it is closest to. When this procedure is completed, the method is recursively applied to all n 'node collections'. This continues until the node collections become small enough such that linear search becomes feasible.

Searching through these access trees (the actual GNAT-search) is then fairly straightforward. The target gate is first compared to all top-level node matrices. We then pick the one that's closest and recursively perform the GNAT-search on the node's subtree. This continues until the search hits a leaf node, where a linear search of remaining gates is performed.

While this drastically reduces search time, there is at least one potential pitfall. Suppose we have an access tree that has two branches on every level, and name the top two nodes A and B. We search for a gate U. Now it is perfectly possible that U is marginally closer to B than to A, but is very near the 'halfway line'. In this case, it can happen that this region is fairly empty on the side of B, while the region just across the halfway line (nearer to A) is well populated (illustrated in Figure 2.2). We will then find a suboptimal approximation to U. Fortunately, this will usually not cause unacceptable losses in accuracy if we choose a sufficiently large number of nodes per level.

Note that it is far from obvious what a good choice for the typical number of branches n is. A larger number of branches would decrease chances of returning suboptimal solutions and result in a 'shallower' tree, but will also increase search time per level. Also, more branches should diminish the effect of the randomized element of the tree building process. Future work might be done in order to determine suitable values for this parameter.

[3]The reason for this random selection is that determining the optimal 'spread' in this situation is an open problem. Note that a side effect of this randomized nature is that some generated trees will exhibit better search performance than others.

Chapter 3

Topological Quantum Computation

In order to construct a physical realization of a qubit, one needs a certain *two-level quantum system*. This can either be a 'pure' two-level system, or a subsystem of a larger construction that allows for the effective decoupling of a two-level state space. An example of a true two-level system is any spin-$\frac{1}{2}$ particle, such as a confined electron. An electron can be in a superposition of its basis spin-up and spin-down states, which can be mapped to basis states for a single qubit. However, many such systems are very susceptible to *decoherence* effects. This roughly means that the superposition state of the qubit is quickly lost through interaction with its environment, which makes it extremely difficult to perform quantum algorithms with such a system.

Therefore, alternative schemes have been developed which provide resistance to small perturbations, greatly diminishing the problem of decoherence. One such scheme is *topological quantum computation* [8] [14]. It essentially encodes qubit states in topological properties of a system, which are much more robust. There are several candidate systems for topological quantum computation, such as certain fractional quantum Hall states. An fairly recent overview of candidate physical systems can be found in [14].

In this section, we explain the general theory of topological quantum computation, a more detailed explanation of which can be found in [15].

3.1 Anyons

Quantum mechanical particles have an important property called *spin*. It is related to the classical phenomenon of angular momentum, but exhibits behavior that has no classical counterpart. For example, the spin of a particle in three spatial dimensions is *quantized*. Just by using the commutation relations of the three dimensional spin operators, we can prove that particles in a world with three spatial (and one temporal) dimensions can have either integral or half-integral spin. That is, the spin of such a particle is always of the form $\frac{k}{2}, k \in \mathbb{N}$.

In three spatial dimensions, there are two important classes of elementary particles: *bosons* and *fermions*. When two identical bosons are exchanged, the wave function of the complete system does not change. When two identical fermions are exchanged, however, their wave function picks up a negative sign. This is called the *Pauli exclusion principle* [16], which largely explains the electron shell structure of atoms. A deep theorem called the *spin-statistics connection* [17] relates the spin of a particle to its exchange statistics. As a result, all particles with integer spin are bosons, while particles with half-integer spin are fermions.

In two spatial dimensions, however, this is not the case. Instead of three axes of rotation, we now have merely one. This means we also cannot have any commutation relations between spin operators in this context, which already hints at the absence of spin quantization. We

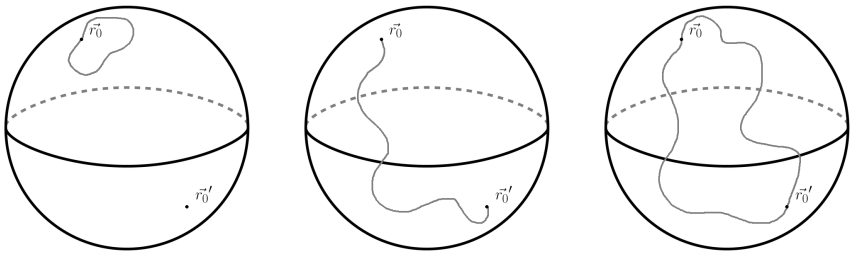


Figure 3.1: Illustration of the possible distinct closed paths on the 2-sphere with opposite points identified. Note that the path in the third image can be continuously shrunk to a point, and is thereby unable to impart a phase shift to the wave function, meaning that it is equivalent to the path in the first figure.

now give a somewhat more rigorous approach to this statement, closely following the reasoning presented in [18].

3.1.1 Exchange statistics in three dimensions

First consider two indistinguishable particles moving in three spatial dimensions. We can form their configuration space by taking all pairs of position vectors (r_1, r_2), effectively creating the space $\mathbb{R}^3 \otimes \mathbb{R}^3$. However, it will turn out to be more convenient to describe their configuration in terms of their center of mass coordinate $(\vec{R})$ and their position relative to eachother $(\vec{r})$. Now we can easily account for their indistinguishability by identifying opposite points in the configuration space for $\vec{r}$ - this can be achieved by dividing it by the group $\mathbb{Z}_2$. Furthermore, we artificially impose the hard-core constraint[1] (preventing the particles from occupying the same position) by removing the origin from the configuration space for $\vec{r}$. The resulting configuration space is

$$\mathbb{R}^3 \otimes \left(\frac{\mathbb{R}^3 - \mathbf{0}}{\mathbb{Z}_2} \right) \tag{3.1}$$

Now we note that the motion of the center of mass is unimportant when considering exchange statistics. Therefore, we only have to consider the configuration space for $\vec{r}$, which we now call S. In order to study the phase change of the wave function when the particles move around each other, we now need to classify all possible closed paths in this configuration space S.

We can make this process somewhat easier by allowing ourselves to visualize it. We can achieve this by restricting the particles to have constant separation distance, so that $|\vec{r}| = c$. The configuration space now becomes a 2-sphere, where diametrically opposite points are identified (through the division by $\mathbb{Z}_2$). Denote this space S_c. A closed path in this space is a function $\phi : [0, 1] \to S_c$ for which $\phi(0) = \phi(1)$. We can now see that there are only two distinct ways of forming a closed path in this space. One is a path that starts out at a point $\vec{r}_0$ and moves in a (possibly deformed) circle back towards itself. This could either be seen as ultimately leaving the particles in place, or as an even number of exchanges. The other is a path that starts at a certain point $\vec{r}_0$ and moves halfway around the sphere (the exact route it takes is unimportant) to its diametrically opposite point, which is again $\vec{r}_0$ by the identification we made above. This represents a single exchange of the two particles. An illustration of three possible paths (of which the first and third are actually equivalent) can be found in Figure 3.1.

The key point is that the former path can always be continuously shrunk to a point, which means that it cannot cause a non-trivial phase shift of the wave function. The latter, however,

[1] It turns out that this can be justified for all particles except bosons. We do not make use of the constraint in the discussion about the exchange statistics, but it will prove to be vital for some aspects of anyon physics, which is why we included it in the description. The justification of the hard-core constraint is detailed in chapter 2 of [18].

cannot be continuously shrunk to a point, implying that it is possible that this imparts a non-trivial phase to the wave function. The exact value η for this phase is easy to compute. To do this, we first note that two succesive exchanges amount to the identity transformation again (we saw this when classifying the possible paths). Suppose the original wave function is $|\psi\rangle$. After one exchange, the wave function becomes $\eta|\psi\rangle$ (where η is a root of unity). After two exchanges, this becomes $\eta^2|\psi\rangle$, which must be equal to the original wave function, which gives us the equation $\eta^2 = 1$, resulting in the values $\eta = \pm 1$. This hints at the fact that these are the only possible phase shifts in a three dimensional system - the phase picked up under one exchange is equal to -1 for fermions and 1 for bosons.

More formally, the argument is based on the *fundamental group* of the relevant configuration space. The fundamental group (or first homotopy group) of a space consists of the group formed by all inequivalent paths (up to homotopy, i.e. paths not continuously deformable to one another) that pass through a certain point in the space. The fundamental group Π_1 of the configuration space in three spatial dimensions is

$$\Pi_1\left(\frac{\mathbb{R}^3 - \mathbf{0}}{\mathbb{Z}_2}\right) = \mathbb{Z}_2 \tag{3.2}$$

So, there are only two distinct classes of paths in this situation, and therefore only two different phases to be picked up. A more detailed discussion of this argument can be found in [19].

3.1.2 Exchange statistics in two dimensions

We now perform the same analysis for two spatial dimensions. The full configuration space is now given by

$$\mathbb{R}^2 \otimes \left(\frac{\mathbb{R}^2 - \mathbf{0}}{\mathbb{Z}_2}\right) \tag{3.3}$$

Again, we ignore the center of mass motion. By the same restrictions, we can picture part of the space by a circle (formally, a 1-sphere). On first sight, everything seems to work exactly as before, but there is one catch. In the three-dimensional case, we saw that two successive exchanges amount to the identity transformation, since it is possible to continuously shrink such a path to a point, all the while remaining in the configuration space. For two dimensions, however, this is not the case. If we now draw a full loop around the circle (representing two successive exchanges), it is impossible to shrink this path to a point (while remaining in the configuration space). Therefore, $\eta^2|\psi\rangle$ does not necessarily have to equal $|\psi\rangle$ anymore. More formally: the fundamental group of the configuration space is isomorphic to the group of integers under addition.

$$\Pi_2\left(\frac{\mathbb{R}^2 - \mathbf{0}}{\mathbb{Z}_2}\right) = \mathbb{Z} \tag{3.4}$$

Interestingly, this opens up a whole new range of possibilities for exchange statistics. Effectively, this introduces a new kind of particle (actually, a quasi-particle, since it only behaves this way in an artificially constructed two-dimensional space) that can have *any* spin. We call such a quasi-particle an *anyon* [20].

We now have a broad range of possibilities for an exchange operator $\mathcal{P}$. It can simply be a root of unity $e^{\frac{2i\pi}{n}}$ - exchange of such particles is always commutative, so we call them *abelian* anyons. However, nothing prevents us from choosing an $n \times n$ unitary matrix for $\mathcal{P}$ (provided that the wave function has n components). Since matrix multiplication isn't necessarily commutative, this allows us to hypothesize quasi-particles for which exchange is *not* commutative. We call these *non-abelian* anyons, and they form the heart of topological quantum computation.

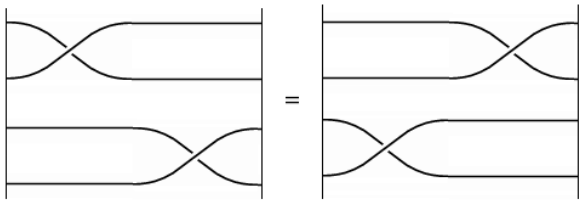


Figure 3.3: Visual representation of commuting braid operations.

3.2 Braiding

We have seen that two successive exchanges of fermions and bosons do not impart a non-trivial phase to the wave function. Therefore, we can characterize the exchange of n such particles by the *permutation group* of order n, denoted S_n. For anyons, this is not the case. We saw that n successive exchanges might very well cause a non-trivial phase shift in the overall wave function, so the permutation group does not accurately represent this behaviour. However, it turns out that a visual representation of anyon exchanges will naturally lead us to a suitable representation.

For this, we consider the *worldlines* of a number of anyons as they undergo time evolution. We can picture these as threads that start at an initial 'time slice' $t = 0$ and end at time slice $t = T$. Since anyons are not allowed to pass through one another, these threads cannot intersect either. Therefore, successively winding two threads 'clockwise' around each other (effectively performing two successive exchanges) is topologically distinct from leaving them in the same place. Figure 3.2 illustrates this concept.

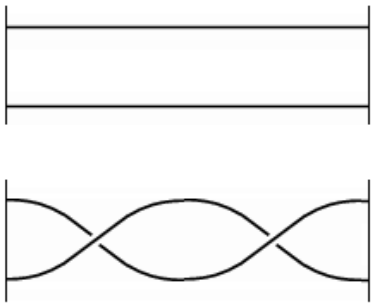


Figure 3.2: Visual representation of the identity operation and two successive counter-clockwise exchanges of two anyons. Since the threads cannot pass through one another, the braids are of distinct topological class.

Now we imagine a system of n anyons and arrange their initial positions on a line. Next, we let $\sigma_i, 1 \leq i \leq n-1$ denote the operation of exchanging particles in positions i and $i+1$ in a counterclockwise manner. Obviously, the inverse σ_i^{-1} is then the operation of performing the same exchange clockwise. Now we note that if we want to perform exchanges on positions that are 'sufficiently far apart', the order of these exchanges is unimportant, i.e. they commute (see Figure 3.3. We can denote this relation

$$\sigma_j \sigma_k = \sigma_k \sigma_j, \quad |j-k| \geq 2 \tag{3.5}$$

However, when we perform successive exchanges on neighbouring strands, the order of operations does matter. By drawing a simple picture we can work out that this relation amounts to

$$\sigma_j \sigma_{j+1} \sigma_j = \sigma_{j+1} \sigma_j \sigma_{j+1}, \quad 1 \leq j \leq n-2 \tag{3.6}$$

This is called the *Yang-Baxter relation* and is pictured in Figure 3.4. It turns out that these two are all relevant relations, and that together they give rise to an infinite group called the *braid group* of order n, denoted B_n [21]. Every anyon model can be described by an irreducible representation of the braid group. Abelian anyons are (irreducibly) described by one-dimensional representations of the braid group, where every generator σ_j has a representation in the form of a root of unity $e^{i\theta_j}$. By the Yang-Baxter relation, we can then derive that $\theta_i = \theta_j$ for all $i, j \leq n-1$, so for abelian anyons of the same type, all exchange phases are equal.

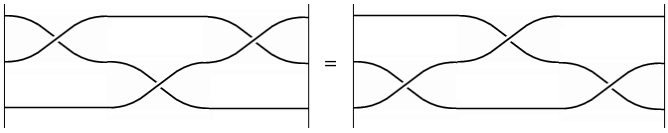


Figure 3.4: Visual representation of the Yang-Baxter relation.

However, there is also an infinite amount of higher-dimensional irreducible representations of the braid group. Such higher-dimensional representations can be non-commuting, leading to a non-abelian anyon model. Such non-abelian representations are particularly interesting, since they might give rise to a model that is *universal* for quantum computation. To see why, recall that such universal models must be able to approximate any operation in $SU(D_n)$ to arbitrary accuracy, where D_n is the dimension of the computational Hilbert space. Formally, a representation of the braid group is a group homomorphism from B_n to $U(V_n)$, where V_n is the topological representation space. Now if the braid group representation is non-abelian, it might be the case that the image of the representation is *dense* in $SU(D_n)$, and can therefore approximate any desired special unitary operation on the computational space. Note that irreducible abelian representations are always one-dimensional, and can therefore never be universal for quantum computation (they are never dense in $SU(D_n)$).

3.3 Fusion

Anyons have a spin-like attribute that differs in nature from the spin we know in three dimensions. We call this attribute the *q-spin* of the anyon. Now nothing prevents us from combining anyons into a composite object, which is again an anyon. This process is known as *fusion*. Different anyon models are distinguished by their so-called *fusion rules*, which dictate how their possible q-spins combine into an 'aggregate' q-spin. A general fusion rule has the form

$$a \times b = \sum_c N_{ab}^c c \tag{3.7}$$

where a and b are the q-spins of the constituent particles and c is the resulting object's q-spin. N_{ab}^c is the number of distinguishable ways in which a charge c can be formed by combining a and b. Note that this fusion process need not be unique: some pairs might be able to fuse in more than one way. If an anyon model contains a fusion rule for which the resulting q-spin is not unique, such a model is non-abelian. Also, the charges are merely 'labels' for the type of anyon we are dealing with - their numerical values are usually not relevant. Therefore, we will often use other characters than numbers, which also helps to prevent confusion about strange-looking (but entirely possible) rules like $2 \times 2 = 1 + 1$.

Anyon models always contain a 'trivial' charge, denoted 1. For any charge a, the equation $1 \times a = a$ holds. Furthermore, an anyon model must be *consistent*, meaning that the process of fusion is associative ($(a \times b) \times c = a \times (b \times c)$), all particles a have an antiparticle ($\exists b : b \times a = 1 + \cdots$), and the braiding and fusion operations must be compatible (we will examine this in

more detail later on). An example of a valid non-abelian anyon model is the one of *Fibonacci anyons*. They can take on q-spins 1 or τ and satisfy the following fusion rules:

$$\begin{aligned} 1 \times 1 &= 1 \\ 1 \times \tau &= \tau \\ \tau \times \tau &= 1 + \tau \end{aligned} \tag{3.8}$$

We can easily see where the name comes from. Suppose we have n such anyons with unknown q-spin. Now we fuse them all together and compute the number of distinguishable ways in which this can happen. All particles with q-spin 1 do not contribute to this amount, so suppose all particles have q-spin τ. Now $\tau \times \tau = 1 + \tau$, so that $\tau \times \tau \times \tau = \tau^3 = \tau + (1 + \tau)$ and $\tau^4 = (1 + \tau) + \tau + (1 + \tau)$. We see that for n particles, the number N_n of different ways they can fuse equals the recursion relation $N_n = N_{n-1} + N_{n-2}$. Furthermore, $N_1 = 0$ and $N_2 = 1$, which gives rise to the sequence of *Fibonacci numbers*. Therefore, the fusion of n Fibonacci anyons can occur in F_n different ways, where F_n is the n-th Fibonacci number.

3.4 Anyon braid matrices

We can denote the Hilbert space of two fused anyons by V_{ab}^c, where again a and b are the q-spins of the constituent particles and c is the resulting q-spin. Its dimension is equal to N_{ab}^c (as above). If we now exchange the two constituent particles, their Hilbert space is denoted V_{ba}^c and we can denote the natural map between these these spaces by the *braid operator* R:

$$R : V_{ab}^c \to V_{ba}^c \tag{3.9}$$

Such a map can be expressed as a unitary matrix, which we often call the R-matrix. This R-matrix might well impart a non-trivial phase to the fusion state (a vector in the N_{ab}^c-dimensional topological vector space V_{ba}^c). We denote the operator that exchanges particles labeled a and b as R_{ab}^c, where c is the combined q-spin.

Furthermore, fusion of anyons is associative - it does not matter whether we combine a and b first and fuse the result with c, or that we fuse b and c first and then fuse with a. Suppose we fuse three anyons a, b, and c. Then we see that we can form the total fusion space V_{abc}^d in two ways: either by first finding the fusion space of a and b and then combining with c, or by first fusing b and c and fusing the resulting space with the one of a. We can describe this relation as follows:

$$V_{abc}^d \cong \bigoplus_e V_{ab}^e \otimes V_{eb}^d \cong \bigoplus_{e'} V_{ae'}^d \otimes V_{bc}^{e'} \tag{3.10}$$

This implies that there are two 'natural' orthornormal bases to describe the total fusion space. We can relate these bases by an appropriate basis transformation, which is again a unitary transformation that we can represent by a matrix. We call this the F-matrix and the operation it performs an *F-move*. We denote it F_{abc}^d.

Now it is not hard to see that these R- and F-matrices must satisfy some constraints. After all, one could construct different compositions of braidings and F-moves that lead to the same transformation of the Hilbert space. These constraints lead to the so-called *pentagon-* and *hexagon-equations*, named after their diagrammatic representations. Figures 3.5 and 3.6 show the diagrams from which the pentagon- and hexagon-equations can be derived.

It turns out that these equations are the only constraints that we need to put in place [22] [23]. As input, we only need the defining fusion rules for a specific anyon model, and we get the relevant R- and F-matrices as a result. When these have been found, we can work out all braid matrices for the model. To see this, consider a system of three anyons. We describe the braiding of the first and second anyon (σ_1, by earlier standards) by the operator R_{ab}^c. If we want to perform a braiding of the second and third anyon, however, we first have to apply a basis transformation by using the F-matrix before applying the braiding operator, so the

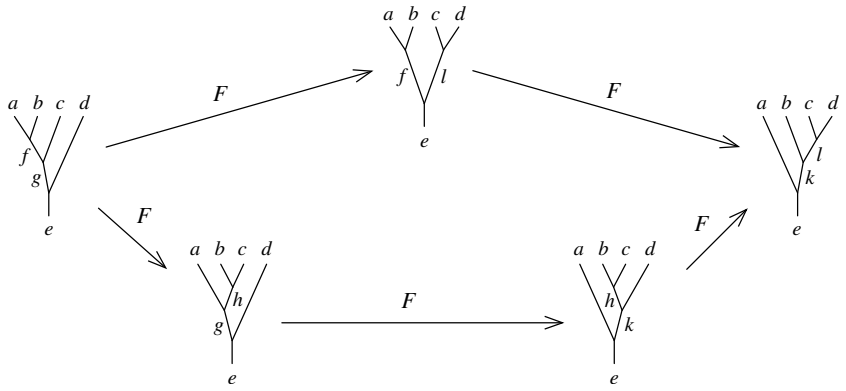


Figure 3.5: Diagrammatic representation of the pentagon-relation that must be satisfied by the R- and F-matrices.

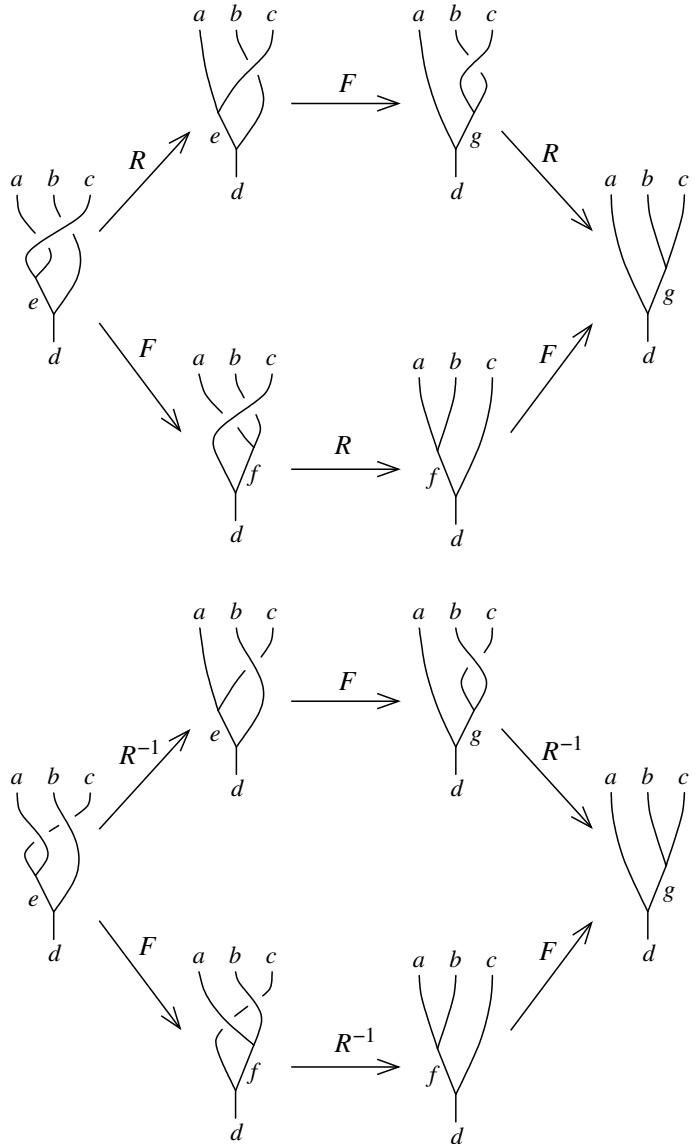
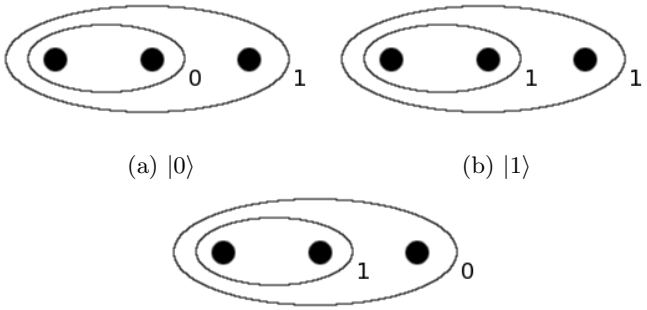


Figure 3.6: Diagrammatic representations of the hexagon-relations.



(a) $|0\rangle$ (b) $|1\rangle$

(c) $|NC\rangle$

Figure 3.7: The mapping of possible anyon system basis states to the qubit and non-computational subspaces.

second elementary braiding operator can be expressed as $\sigma_2 = F^{-1}RF$. By similar reasoning, all the braiding operators $\sigma_i, 1 \leq i < n$ for n quasi-particles can be obtained.

3.4.1 Fibonacci anyons as qubits

In order to let a system of Fibonacci anyons function as a qubit, we must be able to map our two basis states (for example, $|0\rangle$ and $|1\rangle$) onto the system state. A straightforward way to do this uses three Fibonacci anyons and was first proposed in [24]. Let all these anyons have q-spin τ, and consider the first two as a composite anyon. There are now three possible states, two of which we map to the qubit basis states as follows:

The state $|NC\rangle$ is called a *non-computational* state - it does not represent the state of the qubit and we must be careful to avoid transitions into this state. Such a transition is called a *leakage error*. Fortunately, we will see that we can operate on our qubits without leakage in this model.

We now let the first elementary braiding operation σ_1 represent the counter-clockwise exchange of the first two quasi-particles. Obviously, σ_2 then represents the braiding of the second and third quasi-particle, effectively 'interchanging' the second constituent of the composite object with the quasi-particle outside it. It is now possible to work out the pentagon- and hexagon-equations in order to find the F- and R-matrices, which then lead us to the matrix representations of the elementary braiding operations:

$$
\begin{aligned}
R &= \begin{pmatrix} e^{-i4\pi/5} & 0 & 0 \\ 0 & e^{i3\pi/5} & 0 \\ 0 & 0 & e^{i3\pi/5} \end{pmatrix} \\
F &= \begin{pmatrix} \tau & \sqrt{\tau} & 0 \\ \sqrt{\tau} & -\tau & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
\sigma_1 = R &= \begin{pmatrix} e^{-i4\pi/5} & 0 & 0 \\ 0 & e^{i3\pi/5} & 0 \\ 0 & 0 & e^{i3\pi/5} \end{pmatrix} \\
\sigma_2 = F^{-1}RF &= \begin{pmatrix} -\tau e^{-i\pi/5} & \sqrt{\tau}e^{-i3\pi/5} & 0 \\ \sqrt{\tau}e^{-i3\pi/5} & -\tau & 0 \\ 0 & 0 & e^{i3\pi/5} \end{pmatrix} \\
\tau &= \frac{\sqrt{5}-1}{2}
\end{aligned}
\tag{3.11}
$$

Now the upper left 2×2-block acts on the computational subspace spanned by $|0\rangle$ and $|1\rangle$. Since all remaining off-diagonal elements are 0, we see that no leakage into the non-computational space will occur as long as the initial state is fully 'contained' in the computational subspace.

It has been proven that the image of this representation is indeed dense in $SU(2)$ [24]. It even turns out that the model can be extended to n anyons (and therefore, $n/3$ qubits in this qubit state mapping), resulting in a representation of which the image is dense in higher-dimensional versions of the special unitary group. Therefore, Fibonacci anyons can in principle indeed be used for universal quantum computation.

Chapter 4

Results of computational analysis

As a part of this research project, several tools have been created that aid in performing quantum compilation. Several of these procedures can be put together in order to 'compile' arbitrary single-qubit operations in terms of a base set of (fault-tolerant) gates, provided that it generates a dense set in $SU(2)$. The central element of this toolchain is an implementation of the Solovay-Kitaev algorithm. We will discuss all relevant elements of this toolset in detail, along with some specific results or applications, and describe how an end-user would utilize these tools in order to perform compilation. The described procedures are largely based on work previously done by Hormozi, Bonesteel, Zikos, and Simon [25].

A separate procedure exists for analyzing base sets that are suspected to generate a finite subset of $SU(2)$. This could prove to be useful when trying to find out what operations can still be performed fault-tolerantly when making use of these models.

All implementations are written in the MATLAB language. The choice for this platform was mainly motivated by its relatively reliable and speedy handling of numerical processing, along with its high ease of use. When the project has reached some satisfactory level of maturity, it would be possible (and beneficial) to port the code to C or FORTRAN for optimizing performance. However, for our proof-of-concept purposes, this is not required.

Furthermore, we mention some results obtained while applying the toolset to the Fibonacci-anyon braid matrices. Since this theory is proven to provide universal fault-tolerant quantum computation, it is a good test case for the process of compilation. The braid matrices of this model were given in eq. (3.11). The upper 2×2-block in these matrices does not yet have determinant 1 (and is therefore not in $SU(2)$), but we can divide the matrices by the square root of the sub-block determinant in order to bring them into $SU(2)$, as desired.

We also mention some results about the non-universal anyon theory $SU(2)_4$ in order to showcase the procedure for analyzing finite models.

4.1 Generating a gate net

As we have seen in Section 2.5, the Solovay-Kitaev algorithm needs an ϵ-net of gates as input. We can generate such a net by using the method proposed in Section 2.4: we walk the full tree of matrix products up to a feasible depth and 'record' all these products, along with their associated words. If we do this 'naively', i.e. without performing any filtering, the size of this collection will grow exponentially in relation to the walking depth.

However, we generally expect many 'trivial relations' (matrix products that amount to the identity) to come up in this tree. Since the Solovay-Kitaev algorithm requires the input set to contain inverses for all elements, every node will have at least one child node that will evaluate to a previously encountered element. If we incorporate a simple statement that checks whether the current node is an inverse of its parent, all these redundant subtrees will be blocked. Note that we do not have to compare matrices for this: if we keep track of labels, we can perform this check by a simple string comparison. For example, suppose we arrive at a node with word

$aba^{-1}b$. Now we can immediately prevent the evaluation of the subtree attached to its child node b^{-1}.

While this does not remove all redundancy from the tree, it seems to provide the best trade-off between time spent checking and time won by ignoring redundant subtrees. An unfortunate drawback is that this remaining redundancy causes another bottleneck: the program will run out of working memory within a matter of minutes (typically around 10-15), so this becomes the new limiting factor.

Note that the entire procedure is highly parallelizable: the only data that must be shared across threads is the input base set, and only needs read access. It is even an example of an 'embarrassingly parallel' problem: the entire computation of a single subtree is completely independent from its siblings [26].

4.1.1 Results

When the procedure is applied to the braid matrices of the Fibonacci-anyon model, generating the set of gates with words up to length 12 becomes very feasible on an average consumer computer (even without parallelism). The full run takes approximatly ten minutes. From this point on, the limiting factor turns out to be the available working memory.

Upon completion, this results in a collection of roughly 3×10^7 matrices. Such a collection typically yields approximations of $\epsilon < 0.08$ for arbitrary single qubit gates.

A graphical representation of the nets G_3, G_4, G_5, G_7 can be seen in Figure 4.1. Note that this quickly seems to cover the space fairly densely. However, this projection is not a fail-safe method of inspecting the actual covering: it is entirely possible that all points we see merely exist on one 'side' of the 3-sphere, leaving the other side blank. In the depicted case (Fibonacci anyon braid matrices) this concern is unwarranted, as they are known to be universal for quantum computation. If we, however, want to inspect more obscure models, we will have to resort to some different graphical representations - a possible candidate would be a tomographical look at different 'slices' of the space.

4.2 Searching through a net

Searching a collection of matrices for the best approximation to an arbitrary matrix U quickly becomes an expensive operation when the collection grows larger and the search is performed linearly. This is because the linear search generally has to be *exhaustive*: we can only guarantee to have found the best approximation by comparing the search target to all individual elements. There is one case in which this does not hold: it might happen that the desired gate is present in the generated collection (up to machine precision), but this is highly unlikely.

Especially when taking into consideration that the final Solovay-Kitaev algorithm has to perform this search numerous times when we want to execute it to a reasonable depth, it is apparent that optimizing this procedure can cause large speedups. The GNAT-search proposed in Section 2.6.1 turns out to perform spectacularly better than the linear version. This comes at the cost of a long pre-processing time, but the pre-processing step only has to be performed once, so hardly causes any inconvenience.

We already mentioned that this access tree generating procedure involves some decisions based on random numbers, so that some resulting trees will perform better than others. Therefore, if we want to make a valid comparison between a linear search and the GNAT-implementation, we have to include a number of different trees (labeled T_k).

Below, we list the running times for the linear and GNAT-search procedures, when queried for some typical quantum gates. The search space used consists of all Fibonacci-anyon braid matrix products with words up to length 9.

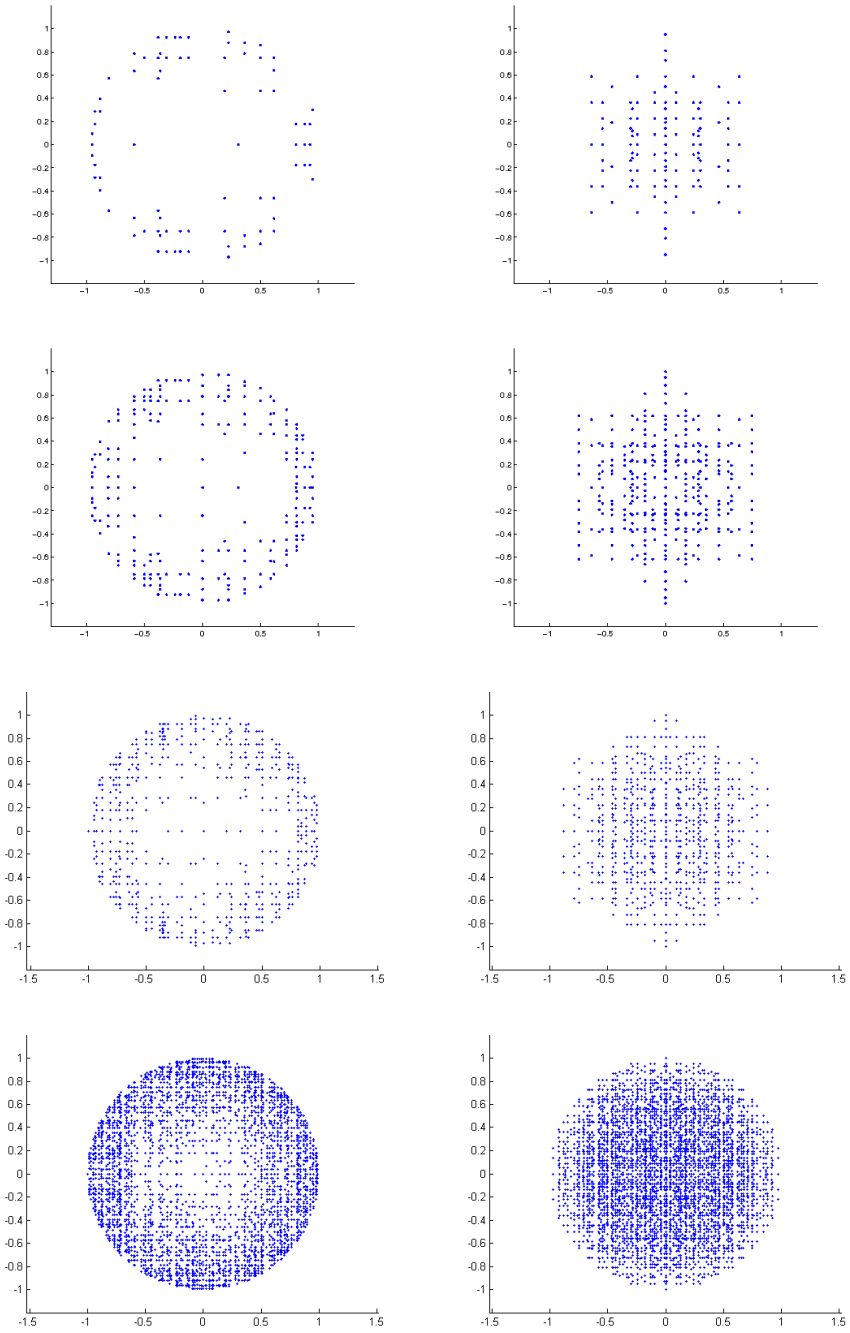


Figure 4.1: A projection of the 4-vectors of every matrix in the sets G_3, G_4, G_5, G_7 respectively (w.r.t. Fibonacci-anyon braid matrices). The 4-vectors of the matrices are obtained through the natural homomorphism from $SU(2)$ to the 3-sphere. The projection in the first image is $(x_1, x_2, x_3, x_4) \mapsto (x_1, x_2)$ and the projection in the second image is $(x_1, x_2, x_3, x_4) \mapsto (x_3, x_4)$.

	H	$\pi/8$	X	Y	Z
G_9	12.06	12.49	11.83	11.84	11.86
T_1	0.010	0.010	0.009	0.015	0.007
T_2	0.010	0.011	0.009	0.023	0.010
T_3	0.010	0.011	0.021	0.009	0.010
T_4	0.010	0.010	0.013	0.011	0.010

Table 4.1: Table listing the running times in seconds of a linear search and the GNAT-implementation. Four different access trees were used in order to demonstrate the discrepancies that arise from their randomized nature. The used trees each have 10 branches per node.

We mentioned that the GNAT-search occasionally returns a suboptimal approximation to the target gate. Therefore, we also list the approximation accuracies of the same searches in the table below.

	H	$\pi/8$	X	Y	Z
G_9	0.066	0.056	0.068	0.113	0.00
T_1	0.066	0.056	0.113	0.113	0.00
T_2	0.066	0.056	0.068	0.113	0.00
T_3	0.066	0.075	0.068	0.113	0.00
T_4	0.066	0.075	0.068	0.113	0.00

Table 4.2: Table listing the approximation error as defined in (2.2) of the result of a linear search and the GNAT-implementation.

It is very clear that the GNAT-search offers a great advantage over the linear implementation. Even though more accurate benchmarks could be performed (these results were obtained by just a single run), we certainly do not have to doubt the effectiveness. Also, a future research direction might be to determine the optimal amount of branches per node. The used access trees have a maximum of ten branches per node. However, taking into consideration the accuracy requirements of the Solovay-Kitaev algorithm, we see that it is not a top priority to improve upon these trees significantly. Also, it turns out that the Z-gate is present in the collection (it is equal to σ_1^5). Therefore, we will omit it in the following compilations.

4.3 Solovay-Kitaev compilation

After performing the aforementioned pre-processing steps, we are ready to run the Solovay-Kitaev algorithm and obtain compiled versions of arbitrary single-qubit gates. We take the same gates as in the search-benchmark as targets. In order to see the enhancements to the accuracy, we list the results for all depths up to 7. The implementation used for this benchmark includes the GNAT-search procedure.

	H	$\pi/8$	X	Y
0	0.06605359392	0.05548890284	0.06821751977	0.11276612186
1	0.00581414836	0.00717664576	0.00558423768	0.00406119681
2	0.00149922151	0.00340729113	0.00158886925	0.00143062834
3	0.00039067005	0.00091699590	0.00031099670	0.00007022803
4	0.00001031980	0.00001600491	0.00001379887	0.00000523950
5	0.00000011750	0.00000009717	0.00000023359	0.00000015841
6	0.00000000022	0.00000000056	0.00000000008	0.00000000097
7	0.00000000022	0.00000000056	0.00000000008	0.00000000010

Table 4.3: Table listing the approximation error for the result of a depth-n Solovay-Kitaev compilation of a number of standard gates.

	H	$\pi/8$	X	Y
0	9	9	9	9
1	48	48	50	50
2	240	240	250	242
3	1196	1212	1202	1198
4	5828	5880	5970	5958
5	29412	29440	29530	29726
6	148532	147096	146210	148262
7	773532	772096	741234	730774

Table 4.4: Table listing the word length for the result of a depth-n Solovay-Kitaev compilation of a number of standard gates.

	H	$\pi/8$	X	Y
0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0
2	0.1	0.1	0.1	0.1
3	0.4	0.4	0.4	0.4
4	1.6	1.6	1.6	1.6
5	6.9	6.9	6.9	6.9
6	32.0	32.0	32.0	32.0
7	158.0	158.0	158.0	158.0

Table 4.5: Table listing the running time in seconds of a depth-n Solovay-Kitaev compilation of a number of standard gates.

We see that the accuracy of the approximation does not seem to improve anymore after six iterations. This is most likely caused by the limited machine precision, and this barrier could therefore be circumvented by performing the calculations in a higher-precision environment.

Furthermore, it turns out that for higher depths, most time ($\sim 85\%$) is spent in a specific subroutine that forms the actual word string of the compiled gate. It is very likely that this routine could be significantly sped up, which will result in large performance gains for the full algorithm. Therefore, it would be advantageous to optimize this method in future versions of the toolset. It would also be interesting to study the difference in performance when using input nets of different maximal word length. For example, if the computations show that G_8 as input net produces very similar results as G_{11}, then we can cut down on pre-processing time in the future. However, this is unlikely, as we have seen that the net tightens quickly as the tree depth increases.

4.4 Analysis of non-universal model

If a certain anyon model is expected to be non-universal for quantum computation, we can compute all distinct braid matrix products by the methods described in Section A.6. We perform this procedure for the anyon model $SU(2)_4$, where we restrict our view to 'integer q-spin' values 1, 2, and 3. Again, 1 acts as the trivial charge. The non-trivial fusion rules then become

$$\begin{aligned} 2 \times 2 &= 1 + 2 + 3 \\ 2 \times 3 &= 2 \\ 3 \times 3 &= 1 \end{aligned} \tag{4.1}$$

We now consider a three-anyon system, where we require the total charge to equal 3. Next, we find the F- and R-matrices for this model and use them to compute the two braid matrices.

$$
\begin{aligned}
R &= \begin{pmatrix} e^{-2i\pi/3} & 0 & 0 \\ 0 & e^{2i\pi/3} & 0 \\ 0 & 0 & e^{i\pi/3} \end{pmatrix} \\
F &= \frac{1}{2}\begin{pmatrix} 1 & -\sqrt{2} & 1 \\ -\sqrt{2} & 0 & \sqrt{2} \\ 1 & \sqrt{2} & 1 \end{pmatrix} \\
\sigma_1 &= R = \begin{pmatrix} e^{-2i\pi/3} & 0 & 0 \\ 0 & e^{2i\pi/3} & 0 \\ 0 & 0 & e^{i\pi/3} \end{pmatrix} \\
\sigma_2 &= F^{-1}RF = \frac{1}{2}\begin{pmatrix} e^{2i\pi/3} & \sqrt{2}e^{i\pi/3} & -e^{2i\pi/3} \\ e^{i\pi/3} & 0 & \sqrt{2}e^{i\pi/3} \\ -e^{2i\pi/3} & \sqrt{2}e^{i\pi/3} & e^{2i\pi/3} \end{pmatrix}
\end{aligned}
\tag{4.2}
$$

The result of walking the tree of matrix products w.r.t. this base set (including the inverses σ_1^{-1} and σ_2^{-1}) is a collection of 162 matrices. However, not all of these have determinant 1. By selecting the subcollection of all matrices with determinant 1, we are left with 27 matrices. Note that this collection forms a group under matrix multiplication, and that this group is a finite subgroup of $SU(3)$. Since there are only five distinct groups (up to isomorphism) of order 27, it is now fairly easy to determine which of these groups it is a faithful representation of.

Upon closer inspection, we see that the group is non-abelian. Since only two of the groups of order 27 are non-abelian, this already eliminates the three other possibilities. Furthermore, al elements turn out to have order 3. Since one of the two remaining groups has an element of order 9, we can eliminate this possibility as well. The only remaining possibility is the *Burnside group* $B(2,3)$. It is the quotient of the free group of two generators and its normal sugroup generated by all third powers of the elements. The tree of group products is displayed in Figure 4.2.

It would be interesting to generalize this for $SU(2)_4$ models with more particles, and for more classes of anyon models. It turns out that there are some connections between knot theory and Burnside groups [27]. Since knot theory and braid groups are closely related, it is possible that these connections might lead to more insight about these anyon models. It is a very interesting direction for future research to work out these connections in more detail, in order to form a more general understanding of $SU(2)_k$ anyon models.

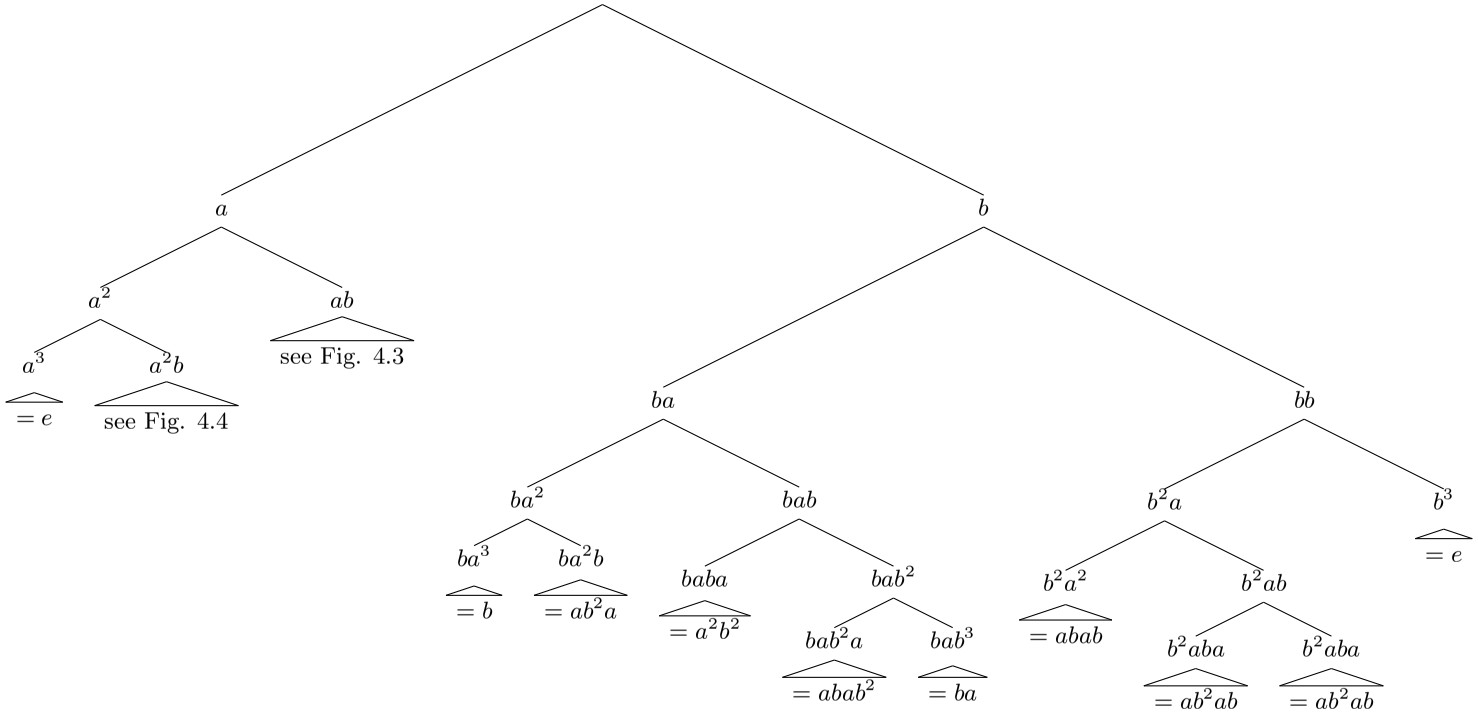


Figure 4.2: Tree of Burnside group $B(2,3)$.

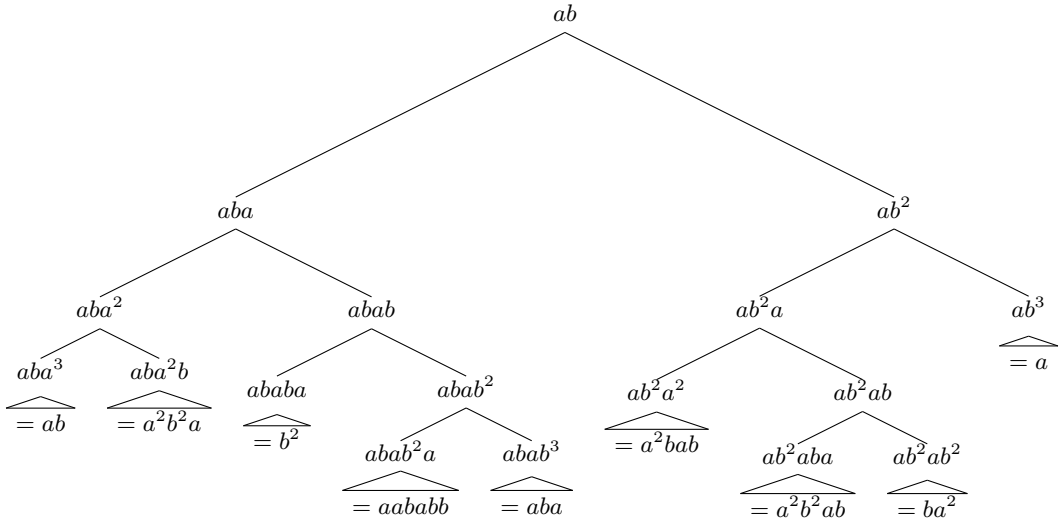


Figure 4.3: Subtree of Burnside group element ab.

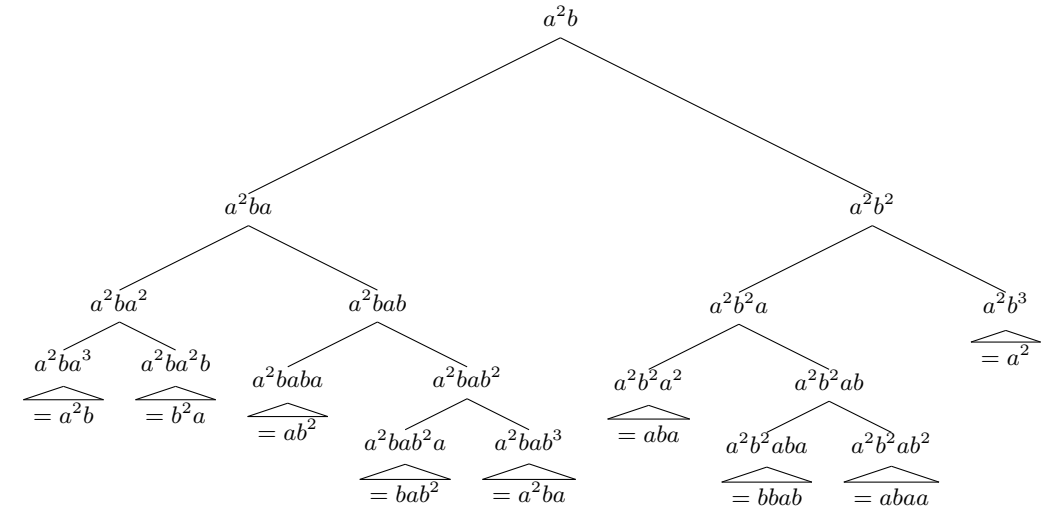


Figure 4.4: Subtree of Burnside group element a^2b.

Appendix A

Description of computational tools

This appendix describes the usage and inner workings of the MATLAB-scripts that have been created as part of this research project. Some of these procedures require the output of one or more other procedures as input. If this is the case, it will be clearly stated. A repository of the code can be found at `http://github.com/JorenB/quantum-compiling`.

For convenience, some widely used constants are stored in the file `constants.m`. For example, it contains a few standard single-qubit operation matrices (for comparison to compiled versions), and can hold a cell array containing the base set of primitive braid matrices (to be used as default input by several of the following procedures).

A.1 Generating a net

The procedure `buildGateNet` takes as input a base set of unitary matrices, containing the representations of available primitive operations (when considering a specific fault-tolerant model) and their inverses. It is required that the first n matrices are the different primitive operations, and that the last n matrices are their respective inverses, in the same order. From this, it generates the tree of possible matrix products with respect to this base set, up to a given depth. If this net is to be used for compilation of arbitrary single-qubit gates, it is clearly preferable that the used model is known to be universal for quantum computation (i.e. the image of its representation is dense in $SU(2)$).

The function can be called with a minimum of one and a maximum of five parameters:

- `depth`: the maximum tree depth to walk (i.e. generate all words up to length `depth`).
- `base`: the base set of matrices, together in a cell array.
- `upper_gates`: mainly used in recursive calls to specify what unique matrices have been generated in the level 'above' the current. Therefore, in standard usage passing a cell array containing just the identity matrix will suffice.
- `upper_words`: similar to `upper_gates`. Passing a cell array containing just an empty string will suffice.
- `full_gates`: mainly used in recursive calls, holds the entire collection of previously generated matrices. As in `upper_gates`, passing a cell array containing just the identity matrix will suffice.

The function returns an array `[total_gates, total_words]`, where `total_gates` is a cell array containing all generated (unique) matrices and `total_words` a cell array containing all

corresponding words. The result is best stored in a cell array `total_gates, total_words`. In this form, it can be directly passed into this implementation of the Solovay-Kitaev algorithm.

If the last three arguments are omitted, the function defaults to the 'standard' inputs, as specified in the above list. If the second argument is omitted as well, the function uses the collection stored in the variable `constants.MATRICES`.

Every time the algorithm finishes computing all products at a certain depth, it outputs the number of levels left to go, the number of new unique matrices found and the time taken at the current level.

A.1.1 Inner workings

The function is designed to call itself recursively. Essentially, it multiplies all matrices supplied in `upper_gates` by all matrices from the base collection `base`. In the meantime, it keeps track of the word by which all matrices are obtained. The matrices and words are stored in `new_gates` and `new_words`. When this process has completed, it calls itself again, passing `new_gates` and `new_words` as values for parameters `upper_gates` and `upper_words`. Also, it lowers the parameter `depth` by one. When `depth` reaches 0, execution stops and the complete collection of computed gates and associated words is returned. We could also say that running the function up to depth n generates all matrices with words up to length n, denoted G_n.

The method includes one simple heuristic to prevent a large amount of redundancy is this 'tree-walking' process. Since the base set contains its own inverses, the product of a matrix with its inverse does not have to be computed and we can ignore the entire subtree below such nodes. It is required that the correct order for matrices and their inverses is maintained, since the algorithm checks for inverses by comparing labels (which is much faster). Note that this only ignores part of the redundant branches, but for our purposes, it will suffice.

A.2 Linear search

We can perform an exhaustive search over a generated net by using the function `linearSearch`. It takes as input a target gate, a collection of gates, and a collection of words, as generated by `buildGateNet`. More precisely:

- `gate`: an arbitrary target gate (must be in $SU(2)$) to be searched for.
- `gates`: a collection of gates as generated by `buildGateNet`.
- `words`: a collection of words as generated by `buildGateNet`. Obviously, it should 'match' the collection of gates.

The function returns the matrix and the associated word of the closest approximation that is available in the search space.

A.2.1 Inner workings

The function compares the target `gate` to all matrices in the search collection. When the entire collection has been looped through, it returns the matrix that had smallest trace distance to the matrix `gate`, along with its word.

A.3 Building an access tree

When a net has been generated, it can be transformed into an access tree structure as described in 2.6.1. This can be done by using the function `gnat`. It takes as input a collection of gates and a collection of words, and a 'branch number', which defines how many branches every node should have. More precisely, its arguments are

- `gates`: a collection of gates as generated by `buildGateNet`.
- `words`: a collection of words as generated by `buildGateNet`. Obviously, it should 'match' the collection of gates.
- `n_branch`: the number of branches per node.

The function returns an access tree structure that can be used in the GNAT-search procedure.

A.3.1 Inner workings

The function first generates `n_branch` distinct random numbers, and uses these to choose the initial node gates. It then compares all remaining matrices to these `n_branch` node matrices and assigns them to a collection 'attached' to the node it is closest to. When this completes, the function calls itself for every such subcollection, resulting in a recursive procedure. The recursion finishes when the size of a node collection drops below 50 - the node then becomes a leaf node. A linear search through 50 matrices is sufficiently fast and it prevents the tree from becoming too deep. Also, it cuts down on pre-processing time.

A.4 Access tree search

When an access tree has been generated, a fast access tree search can be performed over the search space, instead of the slow exhaustive search. This is accomplished with the function `gnatSearch`. It takes as input an arbitrary target $SU(2)$-gate to be searched for, and an access tree structure as generated by `gnat`.

- `gate`: an arbitrary target gate (must be in $SU(2)$) to be searched for.
- `gnat`: an access tree structure as generated by the function `gnat`.

The function returns the matrix and the associated word of the best found approximation to `gate` in the access tree `gnat`.

A.4.1 Inner workings

At every level, the target `gate` is compared to all node matrices. The search is then recursively performed for the subtree of the node that `gate` is closest to.

A.5 The Solovay-Kitaev algorithm

When a net has been generated, the user is ready to run the Solovay-Kitaev algorithm. For better performance, the user also builds the access tree structure and uses the access tree search implementation of the Solovay-Kitaev algorithm, but this is not required.

The standard (non-access-tree-search) implementation of the algorithm can be executed by calling the function `SolovayKitaev`, with the following three arguments:

- `search`: the unitary gate to search for.
- `depth`: the maximum number of recursive calls to make.
- `net`: the full net of gates generated in the preprocessing step.

The access tree version of the algorithm also takes three parameters, of which only the last is different:

- `search`: the unitary gate to search for.

- `depth`: the maximum number of recursive calls to make.
- `gnat`: an access tree structure as generated by the function `gnat`.

The function returns the matrix and the associated word of the approximation to `search` found by applying the Solovay-Kitaev algorithm to depth `depth`.

The function returns a tuple `[gate word]`. Clearly, `gate` is the computed approximation to `search`, while `word` shows the exact chain of matrix products associated to the approximation. Note that the returned gate is always a matrix in $SU(2)$, even if `search` was a matrix with determinant not equal to 1. If the base set of matrices is not special unitary, recomputing the gate making use of the returned `word` w.r.t. this base set will only result in the correct gate up to a certain global phase factor. Using the function `rotateToSU2(gate)`, the user can transform the result to a special unitary matrix.

A.5.1 Inner workings

We follow the approach described in Section 2.5. In short, the first iteration of the algorithm searches for the closest approximation to the input gate `search` in the input `gnat`, and calls this approximation U_0. For the next approximation, it computes the product $UU_0^\dagger$. Since U_0 is 'close' to U and both matrices are unitary, this product will generally be close to the identity. We then decompose this product into the so-called *group commutator* $VWV^\dagger W^\dagger$ with arbitrary matrices $V, W \in SU(2)$. The algorithm then searches in the input `gnat` for the closest approximation to these V, W. When approximations V_0, W_0 have been found, the algorithm is ready to compute the next approximation to U, in the form of $U_1 = V_0 W_0 V_0^\dagger W_0^\dagger U_0$.

For following iterations, the same procedure is repeated in a similar way, with one exception: in the n-th iteration, the search for V, W is replaced by another call to the Solovay-Kitaev algorithm itself, with depth $n-1$. This ensures that sufficiently good approximations to the group commutator decomposition can always be found.

A.6 Generating the full tree of a non-universal model

When a specific set of primitive operations is suspected or known to be non-universal for quantum computation, it is interesting to see what operations this model *can* perform. The function `walkGateTree` computes the entire tree of matrix products w.r.t. a base set and eliminates all subtrees of previously found matrices. Therefore, if the input model is finite, the tree-walking procedure will eventually halt, upon which it returns the full collection. The function takes six parameters, of which only the first one is required. The procedure is closely related to `buildGateNet`, so its parameters are also very similar.

- `depth`: the maximum tree depth to walk (i.e. generate all words up to length `depth`).
- `base`: the base set of matrices, together in a cell array.
- `upper_gates`: mainly used in recursive calls to specify what unique matrices have been generated in the level 'above' the current. Therefore, in standard usage passing a cell array containing just the identity matrix will suffice.
- `upper_words`: similar to `upper_gates`. Passing a cell array containing just an empty string will suffice.
- `full_gates`: used in recursive calls and is returned upon halting. It holds the entire collection of previously generated matrices. As in `upper_gates`, passing a cell array containing just the identity matrix will suffice.
- `full_words`: similar to `full_gates`, holds the associated words for the full matrix collection. Passing a cell array containing just an empty string will suffice.

Again, the last four arguments can be omitted, which lets the procedure use the standard values. If the second parameter is also omitted, it uses the matrices stored in `constants.MATRICES`.

The procedure returns a tuple `[gates words]` which holds the entire collection of generated gates and associated words.

A.6.1 Inner workings

The tree walking procedure is identical to the workings of `buildGateNet`. A major difference is that `walkGateTree` does not make use of the 'trivial relations', but compares every new matrix products to a history collection. If the matrix has not yet been generated before (up to machine precision), it is added to this history collection. If it is a duplicate, it is ignored, and its entire subtree is skipped.

Bibliography

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[2] P. Kaye, R. Laflamme, and M. Mosca, *An Introduction to Quantum Computing*. Oxford University Press, 2007.

[3] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, "Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels," *Physical Review Letters*, vol. 70, no. 13, p. 1895, 1993.

[4] D. Dieks, "Communication by EPR devices," *Physics Letters A*, vol. 92, no. 6, pp. 271–272, 1982.

[5] A. Einstein, B. Podolsky, and N. Rosen, "Can quantum-mechanical description of physical reality be considered complete?," *Physical review*, vol. 47, no. 10, p. 777, 1935.

[6] J. S. Bell *et al.*, "On the Einstein-Podolsky-Rosen paradox," *Physics*, vol. 1, no. 3, pp. 195–200, 1964.

[7] W. H. Zurek, "Decoherence and the transition from quantum to classical–revisited," *arXiv preprint quant-ph/0306072*, 2003.

[8] A. Y. Kitaev, "Fault-tolerant quantum computation by anyons," *Annals of Physics*, vol. 303, no. 1, pp. 2–30, 2003.

[9] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM journal on computing*, vol. 26, no. 5, pp. 1484–1509, 1997.

[10] A. M. Steane, "Error correcting codes in quantum theory," *Physical Review Letters*, vol. 77, no. 5, pp. 793–797, 1996.

[11] A. Y. Kitaev, A. H. Šen&, M. N. Vyalyi, and M. N. Vyalyi, *Classical and quantum computation*, vol. 47. Amer Mathematical Society, 2002.

[12] C. M. Dawson and M. A. Nielsen, "The Solovay-Kitaev algorithm," *Quantum Information and Computation*, 2005.

[13] P. Tien Trung, R. Van Meter, and C. Horsman, "Optimising the Solovay-Kitaev algorithm," *Phys. Rev. A*, 2012.

[14] C. Nayak, S. H. Simon, A. Stern, M. Freedman, and S. D. Sarma, "Non-abelian anyons and topological quantum computation," *Reviews of Modern Physics*, vol. 80, no. 3, p. 1083, 2008.

[15] J. Preskill, "Lecture notes for physics 219: Quantum computation," 2004.

[16] W. Pauli, "Über den zusammenhang des abschlusses der elektronengruppen im atom mit der komplexstruktur der spektren," *Zeitschrift für Physik A Hadrons and Nuclei*, vol. 31, no. 1, pp. 765–783, 1925.

[17] M. Fierz and W. Pauli, "On relativistic wave equations for particles of arbitrary spin in an electromagnetic field," *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 173, no. 953, pp. 211–232, 1939.

[18] S. Rao, "An anyon primer," 1992.

[19] J. M. Leinaas and J. Myrheim, "On the theory of identical particles," *Il Nuovo Cimento B Series 11*, vol. 37, no. 1, pp. 1–23, 1977.

[20] F. Wilczek, "Quantum mechanics of fractional-spin particles," *Physical Review Letters*, vol. 49, no. 14, pp. 957–959, 1982.

[21] J. S. Birman, *Braids, Links, and Mapping Class Groups.(AM-82)*, vol. 82. Princeton University Press, 1974.

[22] S. Mac Lane, *Categories for the working mathematician*, vol. 5. Springer verlag, 1998.

[23] G. Moore and N. Seiberg, "Polynomial equations for rational conformal field theories," *Physics Letters B*, vol. 212, no. 4, pp. 451–460, 1988.

[24] M. H. Freedman, M. Larsen, and Z. Wang, "A modular functor which is universal for quantum computation," *Communications in Mathematical Physics*, vol. 227, no. 3, pp. 605–622, 2002.

[25] L. Hormozi, G. Zikos, N. E. Bonesteel, and S. H. Simon, "Topological quantum compiling," *Physical Review B*, vol. 75, no. 16, p. 165310, 2007.

[26] C. Moler, "Matrix computation on distributed memory multiprocessors," *Hypercube Multiprocessors*, vol. 86, pp. 181–195, 1986.

[27] M. K. Dabkowski and J. H. Przytycki, "Unexpected connections between burnside groups and knot theory," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101, no. 50, pp. 17357–17360, 2004.