Quantum gate learning in engineered qubit networks: quantum half-adder

Liam D. Eloie

Supervised by:

Prof. Sougato Bose *Dr. Leonardo Banchi*

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Theoretical Physics

of

University College London

Department of Physics

University College London

April 10, 2019

I, Liam D. Eloie, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

Quantum gate learning is an up-and-coming field within quantum mechanics that aims to provide a theoretical framework for encoding quantum gates within the unmodulated dynamics of a qubit network, allowing for the operation of quantum gates by simply waiting. The quantum half-adder is a core component for quantum computation and I show that it is possible to encode one within the always-on interactions of a qubit network with only one ancillae qubit by optimising the parameters of the network Hamiltonian using the self-adaptive differential evolution. I was able to find a set of parameters that produced a fidelity between the ideal and implemented gate of 0.98, and that many of the interactions are found to be redundant. This provides us with a gate scheme for the quantum half-adder which could possibly be realised in a superconducting circuit which are known for their fast gate operation times. Additionally, I show that differential evolution can be used to find the parameters required to implement a Toffoli gate within a four qubit network – different from the parameters previously found and suggests flexibility within the implementation of quantum gates using this scheme.

Acknowledgements

I would like to thank my supervisors, Professor Sougato Bose and Dr Leonardo Banchi, for giving me the opportunity to learn about the fascinating field of Quantum Gate Learning and for giving me their invaluable guidance, steering me in the right direction and prompting development throughout my project.

I am eternally grateful to my family and friends; those who have spent their precious time and effort lifting the quality of my life and believing in me every step of the way. En spesiell takk til Martin Lehmann for å hjelpe meg gjennom prosjektet.

And finally, I would like to thank my grandparents, who have supported me an incredible amount throughout my life and have given me the motivation and encouragement to strive.

Contents

Chapter 1

Introduction

Quantum Mechanics, since its discovery, has played an important role in refining our understanding of the universe and supplying us with new techniques applicable across many areas of physics. More recently, the laws of quantum mechanics have been used to explore areas of information technology to create computational devices that allow us to more efficiently run algorithms that are known to be difficult for classical computers [1]. Realisations of quantum computing prove to be a difficult task; quantum states are easily influenced by external noise, resulting in information loss. In the traditional quantum computing circuit model, algorithms are broken down into a series of quantum gates. Unfortunately, to realise these algorithms, it requires a control pulse for every gate in the series – this introduces small errors along the way. In the past couple of years, optimisation techniques have been used for constructing quantum gates within the always-on interactions of a qubit network [2, 3, 4]. This has a benefit over the traditional quantum computing circuit model, as it doesn't require the algorithm to be decomposed into a series of gates – minimising the control pulses needed to perform the algorithm. Instead, computation can be done by simply waiting for it to happen. I propose the question: is it possible to encode more complex quantum gates using this gate learning scheme, specifically, the quantum half-adder? I will be designing a network of qubits using an evolutionary optimisation algorithm–differential evolution–to try and implement the quantum half-adder with an acceptably high fidelity. A half-adder is a fundamental gate that has the ability to perform addition on single binary-digits. The

successful implementation of a quantum half-adder would be an essential step towards the construction of fast, fault-tolerant quantum algorithms as basic arithmetic is at the core of all complex calculations in a computer.

The first section of this thesis will be a comprehensive overview of the theory needed to understand quantum gate learning. It will cover the basics of quantum computing; quantum bits, quantum gates, and what a quantum half-adder is. It will then talk about quantum information; quantum states, time evolution of systems, and how to measure the similarity between two quantum states. Combining these ideas, the next subsection will talk about the framework of quantum gate learning. The last section will cover differential evolution, how it works, and a variant called self-adaptive differential evolution.

The second section will cover the methodology of my project; what I used in order to work with quantum systems, how I designed a quantum half-adder, how to construct a qubit network, and how I implemented differential evolution to encode the quantum half-adder within the qubit network.

The last sections are about my results and any discussion related to those results.

Chapter 2

Background

This section is going to cover the background and theory required to understand the implementation of a quantum half-adder in the unmodulated dynamics of a qubit network. The first topic will discuss the basic principles of quantum computing such as qubits, quantum logic gates, and the definition of a quantum half adder. The second topic will give an overview on quantum information: how we represent quantum states, find the time evolution of a quantum system, measure the similarity between two quantum states using fidelity, and how to vectorise the fidelity measure. The third topic will combine the previous topics together to describe the idea behind quantum gate learning and how to set up the framework in which to implement a quantum gate in the unmodulated dynamics of a qubit network. The last topic will go through differential evolution, the algorithm I will be using to optimise the qubit network.

2.1 Quantum Computing

2.1.1 Quantum Bits

The very basic building blocks of a classical computer boil down into two simple components: bits and logic gates. Logic gates are used to switch the binary state of bits in such a way to perform a certain computation. Borrowing from this idea and using the concept of quantum superposition, the idea of a quantum bit was formed. More commonly known as a qubit, a portmanteau of quantum and bit, this fundamental elementary unit of quantum information along with quantum logic

gates are used to construct the quantum circuit model of quantum computing. A qubit can be described by a two-state quantum system such as a spin-$\frac{1}{2}$ particle [5].

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.1}$$

The advantages of a qubit over a classical bit, which can only exist as either a 0 or 1 and perform one calculation at a time, is that they can exist as a $|0\rangle$, $|1\rangle$ or any linear superposition between the two, allowing the information of multiple calculations at a time to be stored within the probability amplitudes of the quantum state. This can ultimately lead to the exponential speed-up of certain algorithms over their classical counterparts.

Unfortunately, unlike classical states, quantum states are non-deterministic. Upon measuring a quantum state, you will be met with a probabilistic outcome depending on the probability amplitudes. For example, the probability of measuring the outcome of $|0\rangle$ in Equation 2.1 would be $|\alpha|^2$. Likewise, the probability of measuring the outcome of $|1\rangle$ would be $|\beta|^2$. Since an outcome must occur, the sum of the coefficients squared of the quantum state must add up to 1.

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.2}$$

The condition in Equation 2.2 leads to the generalised state of a qubit.

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle \tag{2.3}$$

Equation 2.3 is interesting as it allows the visualisation of a qubit on a unit sphere. This is known as the Bloch Sphere representation and can be seen in Figure 2.1.

2.1.2 Quantum Circuits and Algorithms

Conventional quantum computers are composed of qubits and quantum gates. All quantum mechanical operators must be unitary, quantum gates are no exception. By using these unitary operators, we can manipulate the state of single or multiple

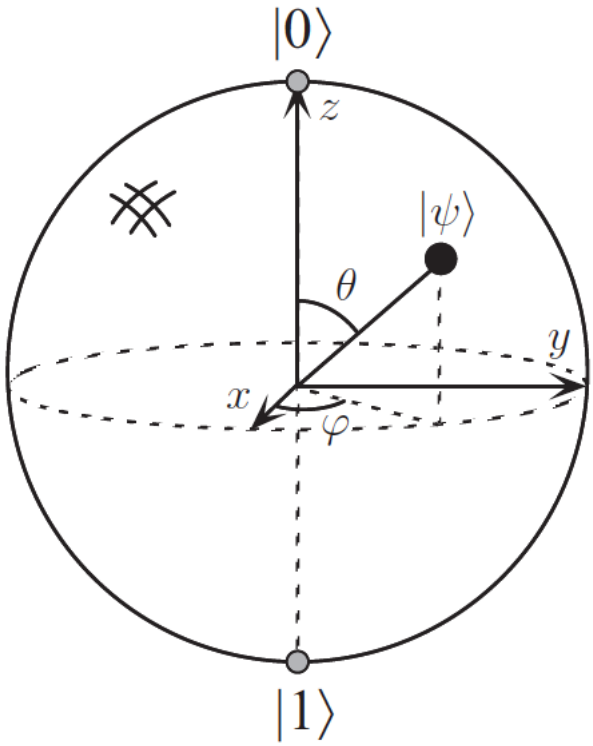


Figure 2.1: Bloch sphere representation of a qubit state [1].

qubits. Going back to the Bloch sphere representation, the operation of a quantum gate to a qubit will result in a rotation around the sphere. There are lots of different elementary quantum gates that perform fundamental transformations on a single qubit state. Amongst the most important are:

- **Hadamard gate**

$$|0\rangle \rightarrow \frac{|0\rangle + |1\rangle}{\sqrt{2}} \qquad\qquad |1\rangle \rightarrow \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

- **Pauli-X**

$$\alpha|0\rangle + \beta|1\rangle \rightarrow \alpha|1\rangle + \beta|0\rangle$$

- **Pauli-Y**

$$\alpha|0\rangle + \beta|1\rangle \rightarrow i\alpha|1\rangle - i\beta|0\rangle$$

- **Pauli-Z**

$$\alpha|0\rangle + \beta|1\rangle \rightarrow \alpha|0\rangle - \beta|1\rangle$$

- **Phase shift**

$$\alpha|0\rangle + \beta|1\rangle \rightarrow \alpha|0\rangle + e^{i\phi}|1\rangle$$

Perhaps more interestingly, there are quantum gates that allow us to conditionally manipulate qubits. These gates work by performing a quantum gate on a target qubit depending on the state of a set of control qubits. Some examples of fundamental multi-qubit quantum gates include:

- **CNOT gate** - acts on two qubits. If the first qubit is $|1\rangle$, it applies Pauli-X on the second qubit. Otherwise, the state is left unchanged.
- **Toffoli gate** - acts on three qubits. If the first two qubits are in the state $|1\rangle$, it applies Pauli-X on the third qubit. Otherwise, the state is left unchanged.
- **Fredkin Gate** - If the first qubit is in the state $|1\rangle$, the last two qubit states are swapped.

These gates can be positioned in a unique series to perform quantum algorithms – this is called a quantum circuit[6]. Owing to the principles of superposition and entanglement, these quantum algorithms are capable of solving mathematical problems that would otherwise be rather difficult to solve for any classical algorithm. Shor's algorithm dealing with integer factorisation and discrete logarithms in polynomial time [7], as opposed to sub-exponential time using the most efficient asymptotic classical algorithms, (e.g. The Number Field Sieve) [8] and Grover's algorithm for searching an unstructured database in only $O(\sqrt{N})$ steps [9] are prime examples of the power and potential of quantum computation.

2.1.3 Quantum Half-Adder

Low-level manipulation of a bit string is easy enough, but how does one get a classical computer to perform more complex calculations? By using a set of elementary logic gates, more complicated structures are able to be built. In information theory,

there exists a digital circuit called an 'adder' which is able to perform addition on two binary numbers. This is fundamental to the workings of a computer, as basic arithmetic is at the core of all complex calculations. An adder can be broken down into a series of half-adders, a simpler structure that has the ability to perform addition on two single binary digits. A classical half-adder takes two bits as input and produces a sum bit and a carry bit as its output. The sum of the two binary bits may be computed by performing a simple XOR gate. Likewise, simply apply an AND gate to the two binary bits to compute the carry as seen in Figure 2.2(a). By

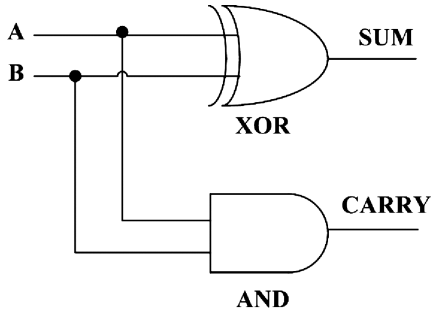


Figure 2.2: (a) A half-adder constructed with a XOR and AND gate[1].

TRUTH TABLE OF HALF-ADDER			
INPUT		OUTPUT	
X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Figure 2.3: (b) The truth table of the half-adder circuit[2].

evaluating the output of the half-adder for every possible input, it is clear how the half-adder performs addition 2.2(b).

Analogous to the construction of classical structures, quantum logic gates can also be combined to achieve a certain computation. The quantum counterpart of the half-adder, known as the quantum half-adder, may be built using two simple multi-

[1]https://nisalsworld.blogspot.co.uk/2016/05/combination-circuits.html
[2]http://www.bscshortnote.com/what-is-half-adder/

qubit logic gates in series acting on a qubit register comprising of three qubits: the Toffoli gate and the CNOT gate represented in Figure 2.4. Acting these gates on

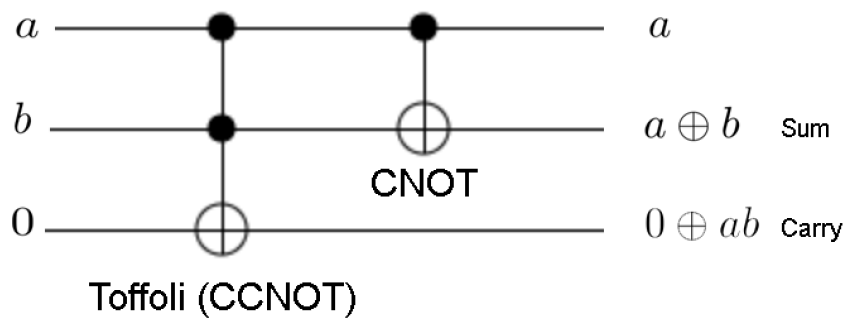


Figure 2.4: A quantum half-adder constructed with a Toffoli and CNOT gate.

the initial state $|a\rangle \otimes |b\rangle \otimes |0\rangle$, working from left to right, the algorithm performs in the following way.

$$\begin{aligned} \hat{C}_{not}^{a,b}\hat{T}(|a\rangle \otimes |b\rangle \otimes |0\rangle) &= (\hat{C}_{not} \otimes \hat{I})(|a\rangle \otimes |b\rangle \otimes |ab \oplus 0\rangle) \\ &= |a\rangle \otimes |a \oplus b\rangle \otimes |ab \oplus 0\rangle \end{aligned} \tag{2.4}$$

where the $\oplus$ symbol means modulo-2 addition. By going through every possible single binary state of a and b, evaluating the final quantum state, and constructing a truth table as seen in Figure 2.5, confirmation that this series of quantum gates performs a half-adder can be shown.

Truth Table of Quantum Half-Adder					
Input			Output		
a	b	c	a	b(sum)	c(carry)
0	0	0	0	0	0
0	1	0	0	1	0
1	0	0	1	1	0
1	1	0	1	0	1

Figure 2.5: The truth table obtained when evaluating the quantum half-adder in Figure 2.4 over the three-qubit binary states with the third qubit set to 0.

Going a step further, since all gates are unitary operators, they may be represented

by unitary matrices. This will be a helpful representation when it comes to simulating the gate on a classical computer.

The matrix representation for the CNOT gate and the Toffoli gate are shown in Equation 2.5 and 2.6, respectively.

$$\hat{C}_{not} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.5}$$

$$\hat{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.6}$$

In order to compute the matrix representation of the quantum half-adder, one can simply substitute Equation 2.5 and Equation 2.6 into the operator of the quantum

half-adder in Equation 2.4.

$$
\begin{aligned}
\hat{G}_{qha} &= \hat{C}_{not}^{a,b}\hat{T} = (\hat{C}_{not} \otimes \hat{I})\hat{T} \\
&= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}
\end{aligned}
\tag{2.7}
$$

2.2 Quantum Information

2.2.1 Density Matrix

Quantum mechanics can be formulated in two different ways: state vector formalism and density operator formalism. The density operator approach to quantum mechanics is great when you don't know the exact overall state of the system, this is done by assigning classical probabilities to each state, these are known as mixed states. When working with the time-evolution of a more complex system, it is extremely useful to switch over to the density operator formalism. This allows us to see how subsystems of the overall system evolve over time.

The density operator can be defined in the following way:

$$\rho = \sum_k p_k |\psi_k\rangle \langle\psi_k|$$

where p_k is the probability of the system existing in the state $|\psi_k\rangle \langle\psi_k|$.

The density operator has a few nice properties,

1. The trace of ρ is always equal to 1, this conditions comes from the fact that probabilities sum to 1.

2. The eigenvalues of ρ are always real-valued. This is because they are representing physical observations.

3. The eigenvalues of ρ are always positive. Probabilities cannot be negative.

Let a quantum system consist of two subspaces, A and B. Their combined quantum state can be represented by the tensor product of their individual states,

$$\rho_{AB} = \rho_A \otimes \rho_B$$

By combining the two subspaces into one quantum state, there is no lost of generality of the individual subspaces. This is one of the main strength of the density operator formalism and can be demonstrated by taking the partial trace of the system over a subspace of choice. Consider extracting the information about ρ_A from the composite density matrix ρ_{AB}, this is known as the reduced density matrix ρ_A.

$$\begin{aligned} \mathrm{Tr}_B(\rho_{AB}) &= \mathrm{Tr}_B(\rho_A \otimes \rho_B) \\ &= \rho_A \otimes \mathrm{Tr}(\rho_B) \\ &= \rho_A \otimes I_B \\ &= \rho_A \end{aligned}$$

where the fact that the trace of a density matrix is always equal to 1 is used.

2.2.2 Time Evolution of a Quantum System

Quantum systems are governed by the Time-Dependent Schrödinger Equation (TDSE) as seen in Equation 2.8.

$$i\hbar\frac{\partial\psi}{\partial t} = -\frac{\hbar^2}{2m}\nabla^2\psi + V(x,y,z)\psi \tag{2.8}$$

Assuming a time-independent Hamiltonian, the solution to this partial differential equation is separable in time and space, and as such lends itself to the following form:

$$\psi(x,y,z,t) = R(x,y,z)T(t) = e^{-\frac{it}{\hbar}\hat{H}}R(x,y,z) = U(t)R(x,y,z)$$

$$U(t) = e^{-\frac{it}{\hbar}\hat{H}} \tag{2.9}$$

where $U(t)$ is known as the unitary operator that governs the time-evolution of the system. From this, it is easy to deduce that the time evolution of a quantum system is solely governed by the time-independent Hamiltonian of the system. That is, given a quantum state $|\psi\rangle$, the form of the quantum state after a time t will be given by,

$$|\psi(t)\rangle = e^{-\frac{it}{\hbar}\hat{H}}|\psi(0)\rangle = U(t)|\psi(0)\rangle$$

This easily extends to the density operator formalism of quantum mechanics and provides a useful way of dealing with the time evolution of more complicated quantum systems.

$$\rho(t) = |\psi(t)\rangle\langle\psi(t)| = e^{-\frac{it}{\hbar}\hat{H}}|\psi(0)\rangle\langle\psi(0)|e^{\frac{it}{\hbar}\hat{H}} = e^{-\frac{it}{\hbar}\hat{H}}\rho(0)e^{\frac{it}{\hbar}\hat{H}} = U(t)\rho(0)U^{\dagger}(t)$$

Therefore, given an initial density matrix, one is able to calculate the time evolution of that density matrix after a certain time t.

2.2.3 Fidelity

In order to compare and distinguish between two quantum states, it is necessary to introduce distance measures. As the name suggests, a distance measure is a measure of how 'far' away one thing is from another. As such, a distance measure must satisfy the properties of a metric. Specifically, a metric must be: positive, symmetric, obey the triangle inequality, and be equal to 0 when the states are the same.

- $D(\rho,\sigma) \geq 0$
- $D(\rho,\sigma) = D(\sigma,\rho)$
- $D(\rho,\sigma) \leq D(\rho,\theta) + D(\theta,\sigma)$
- $D(\rho,\rho) = 0$

In quantum information, there are two main measures used to compare quantum states. The trace distance and the Bures' angle:

$$D(\rho,\sigma) = \frac{1}{2}\mathrm{Tr}|\rho-\sigma| \tag{2.10}$$

$$A(\rho,\sigma) = \cos^{-1}(F(\rho,\sigma)) = \cos^{-1}\left(\mathrm{Tr}\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}\right) \tag{2.11}$$

Although frameworks can be built with both measures, I will mainly be focusing on a measure that is used to calculate the Bures' angle, the fidelity. The fidelity function isn't a metric, but can be used to construct the Bures' angle, which does satisfy the properties of a metric.

The fidelity between two quantum states is defined by Equation 2.12

$$F(\rho,\sigma) = \mathrm{Tr}\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}} \tag{2.12}$$

2.2.3.1 Fidelity of the same state

Using the result from Uhlmann's theorem [10], the value of the fidelity when both input states are the same becomes trivial to calculate.

$$F(\rho,\rho) = \max_{|\psi\rangle,|\psi\rangle} |\langle\psi|\psi\rangle|^2 = \max_{|\psi\rangle,|\psi\rangle} |1|^2 = 1$$

This result confirms the fact that fidelity is not a metric, as it is not equal to zero. This isn't a problem since substituting this value into Equation 2.12 shows that the fidelity angle, is indeed, a metric ($\cos^{-1}(1) = 0$). This demonstrates the fact that the more similar the two quantum states, the closer the fidelity will be to 1.

2.2.3.2 Fidelity between two operators

The fidelity between two operators can be define in the following way,

$$F(\mathscr{E},G) = \langle\psi| G^\dagger \mathscr{E}(\psi) G |\psi\rangle \tag{2.13}$$

2.2.4 Average Gate Fidelity

In quantum computation, trying to implement a quantum gate, G, described by some unitary operation often occurs. Unfortunately, the implementation is usually a slightly noisy version of the gate, resulting in a quantum channel described by the quantum operation, $\mathscr{E}$. A way to test the success of our implementation is to compute a measure of the similarity between the quantum channel and the target unitary operation. The measure used is called the average gate fidelity. Given that our quantum operation $\mathscr{E}$ is trace-preserving, the average fidelity is defined by

$$\bar{F}(\mathscr{E}) = \int d\psi \, \langle\psi| \mathscr{E}(\psi) |\psi\rangle \tag{2.14}$$

where the integral is over the Haar measure $d\psi$ on the state space. The definition can be extended to compute a measure of the similarly between a quantum operation, $\mathscr{E}$, and a quantum gate, G.

$$\bar{F}(\mathscr{E},G) = \int d\psi \, \langle\psi| G^\dagger \mathscr{E}(\psi) G |\psi\rangle \tag{2.15}$$

The closer the result of this integral is to 1, the closer the quantum operation $\mathscr{E}$ represents G.

The average fidelity can be represented in a more computer friendly form, one can show:

$$\bar{F}(\mathscr{E},G) = \frac{1}{D+1} + \frac{1}{D(D+1)} \sum_{i,j,k,l} G^*_{ik} \mathscr{E}_{ijkl} G_{jl} \tag{2.16}$$

where D is the dimension of the D-dimensional Hilbert space, $G_{jl} = \langle q_j | G | q_l \rangle$, $\mathscr{E}_{ijkl} = \langle q_i | \mathscr{E}[|q_k\rangle \langle q_l|] | q_j \rangle$, where $|q\rangle$ are the basis functions that span the D dimensional Hilbert space.

2.2.5 Vectorised Average Gate Fidelity

When it comes to evaluating this form of the average gate fidelity, it requires a lot of computational effort to explicitly run through four summations. It would be nice to find a way to vectorise this expression in order to decrease the amount of time it takes to compute.

By considering the Choi matrix, $\rho_{\mathscr{E}}$, it allows us to vectorise the average gate fidelity.

$$\rho_{\mathscr{E}} = (I \otimes \mathscr{E}) |\Phi\rangle \langle\Phi|$$

where $|\Phi\rangle$ are the maximally entangled quantum states, $\frac{1}{\sqrt{d}} \sum_i |ii\rangle$

By expanding out the Choi matrix and doing some matrix algebra, a useful expression for the quantum channel in terms of the Choi matrix can be found.

$$\begin{aligned} \rho_{\mathscr{E}} &= (I \otimes \mathscr{E}) \frac{1}{\sqrt{D}} \sum_m |mm\rangle \frac{1}{\sqrt{D}} \sum_n \langle nn| \\ &= \frac{I}{D} (I \otimes \mathscr{E}) \sum_{m,n} |mm\rangle \langle nn| \end{aligned}$$

To construct a matrix representation of the Choi matrix, contract both sides of Equation 2.16 by 2D basis states $\langle ki|$ and $|lj\rangle$

$$\begin{aligned}
\langle ki|\,\rho_{\mathscr{E}}\,|lj\rangle &= \frac{1}{D}\langle ki|\,(I\otimes\mathscr{E})\sum_m\sum_n |mm\rangle\langle nn|\,|lj\rangle \\
&= \frac{1}{D}\langle ki|\sum_m\sum_n (|m\rangle\langle n|\otimes\mathscr{E}[|m\rangle\langle n|])\,|lj\rangle \\
&= \frac{1}{D}\sum_m\sum_n \langle k|m\rangle\langle n|l\rangle\otimes\langle i|\,\mathscr{E}[|m\rangle\langle n|]\,|j\rangle \\
&= \frac{1}{D}\sum_m\sum_n \delta_{km}\delta_{nl}\otimes\langle i|\,\mathscr{E}[|m\rangle\langle n|]\,|j\rangle \\
&= \frac{1}{D}\langle i|\,\mathscr{E}[|k\rangle\langle l|]\,|j\rangle \\
&= \frac{1}{D}\mathscr{E}_{ij,kl}
\end{aligned}$$

Rearrange this in terms of $\mathscr{E}_{ij,kl}$ to express the quantum channel in terms of the Choi matrix, $\rho_{\mathscr{E}}$.

$$\mathscr{E}_{ij,kl} = D\,\langle ki|\,\rho_{\mathscr{E}}\,|lj\rangle$$

Using this, the average gate fidelity can be rewritten as,

$$\bar{F}(G,\mathscr{E}) = \frac{1}{D+1} + \frac{1}{D+1}\sum_{i,j,k,l} G^*_{ik}\,\langle ki|\,\rho_{\mathscr{E}}\,|lj\rangle\,G_{jl}$$

By defining $|G\rangle = \sum_{ij} G_{ij}\,|ji\rangle$, the average gate fidelity can be expressed in a concise vectorised form.

$$\bar{F}(G,\mathscr{E}) = \frac{1}{D+1} + \frac{1}{D+1}\,\langle G|\,\rho_{\mathscr{E}}\,|G\rangle$$

This expression is only useful if $\rho_{\mathscr{E}}$ can be expressed in terms of the basis functions and ancillae states. This will require the explicit evaluation of the Choi matrix for the channel $\mathscr{E}$. The way to do this is to introduce indices to denote which Hilbert space each object belongs to, otherwise the computation gets messy and it's easy to

lose track of what operator acts on what Hilbert space.

$$\begin{aligned}
\rho_{\mathscr{E}} &= (I_{1,2,3} \otimes \mathscr{E}_{4,5,6}) |\Phi\rangle \langle\Phi| \\
&= (I_{1,2,3} \otimes \mathscr{E}_{4,5,6}) \frac{1}{D} \sum_{ij} |ii\rangle \langle jj| \\
&= \frac{1}{D} (\sum_{ij} |i\rangle_{1,2,3} \langle j|_{1,2,3} \otimes \sum_{ij} \mathscr{E}_{4,5,6}[|i\rangle_{4,5,6} \langle j|_{4,5,6}]) \\
&= \frac{1}{D} (\sum_{ij} |i\rangle_{1,2,3} \langle j|_{1,2,3} \otimes \sum_{ij} \mathrm{Tr}_7(U_{4,5,6,7} |i\rangle_{4,5,6} \langle j|_{4,5,6} \otimes |\phi\rangle_7) U^{\dagger}_{4,5,6,7}) \\
&= \frac{1}{D} \mathrm{Tr}_7(U_{4,5,6,7} \sum_{ij} |i\rangle_{1,2,3} |i\rangle_{4,5,6} \langle j|_{1,2,3} \langle j|_{4,5,6} \otimes |\phi\rangle_7 \langle\phi|_7 U^{\dagger}_{4,5,6,7})
\end{aligned}$$

It is now possible to define a state vector $|\Xi\rangle$ to rewrite the last expression,

$$|\Xi\rangle = |\Phi\rangle \otimes |\phi_A\rangle = \frac{1}{\sqrt{D}} \sum_i |i\rangle |\phi_A\rangle |i\rangle \tag{2.17}$$

The Choi matrix can now be written in the following way,

$$\rho_{\mathscr{E}} = \mathrm{Tr}_7(U_{4,5,6,7} |\Xi\rangle \langle\Xi| U^{\dagger}_{4,5,6,7})$$

which in terms of register qubits and ancillae, spanning the time-evolution of the system into $H^{\otimes 7}$ Hilbert space, turns out to be

$$\rho_{\mathscr{E}} = \mathrm{Tr}_A((\hat{I}^{\otimes 3} \otimes U) |\Xi\rangle \langle\Xi| (\hat{I}^{\otimes 3} \otimes U)^{\dagger}) \tag{2.18}$$

Finally, substituting this back into the average gate fidelity results in,

$$\bar{F}(G, \mathscr{E}) = \frac{1}{D+1} + \frac{1}{D+1} \langle G| \mathrm{Tr}_A((\hat{I}^{\otimes 3} \otimes U) |\Xi\rangle \langle\Xi| (\hat{I}^{\otimes 3} \otimes U)^{\dagger}) |G\rangle \tag{2.19}$$

2.3 Quantum Gate Learning

In order to implement a quantum gate, a system must be established that can perform such a task. The system I will be considering is a network of qubits that interact with each other via two-body spin-coupling exchange interactions and that

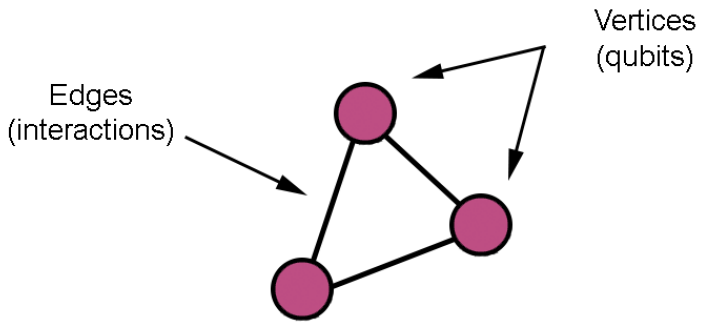


Figure 2.6: A directed graph consisting of vertices and edges.

can evolve over time. The main idea is to optimise these interactions in such a way to encode a desired quantum gate into the unmodulated dynamics of the quantum network using optimisation techniques. This scheme for performing quantum computation has a large benefit over the traditional quantum circuit model–it is no longer required to break down a quantum gate or algorithm into a series of simpler components. When realising quantum computation using the circuit model, sophisticated control pulses are used for operating through the circuit to perform the computation. Each one of these control pulses introduces an error to the computation. Fortunately, the quantum gate learning scheme eliminates these control pulses, avoiding any errors that otherwise would have been induced – allowing quantum computation to happen simply by waiting.

Consider a quantum network consisting of an undirected graph (V, E) of vertices V and links E as can be seen in Figure 2.6. The vertices V are made up of single qubits and the links E correspond to the 2-local spin-coupling interactions between the qubits. The quantum network can be described by the following time-independent Hamiltonian,

$$\mathscr{H} = \sum_{(n,m)\in E} \sum_{\alpha,\beta} J_{nm}^{\alpha\beta} \frac{\sigma_n^\alpha \sigma_m^\beta}{4} + \sum_{n\in V} \sum_{\alpha} h_n^\alpha \frac{\sigma_n^\alpha}{2} \tag{2.20}$$

where σ_n^α, $\alpha = x, y, z$, are the Pauli matrices acting on qubit n, $J_{nm}^{\alpha\beta}$ is the interaction strength between the nth and mth qubit via $\alpha\beta$-spin coupling interaction, and h_n^α is the strength of the external magnetic field on the nth qubit in the α direction.

The idea is to optimise the parameters, J and h, of the time-independent Hamiltonian that governs the time-evolution of our network of qubits in such a way to maximise the fidelity between the time-evolution operator and our desired gate. Quantum gates operate with fixed times, so for simplicity, t will be set to 1. Additionally, to simplify the expression, $\hbar$ will also be set to 1.

It is often undesirable to encode a N-qubit quantum gate G within the time evolution of a N-qubit network, U.

This would be an easy problem to solve, as it would only require solving Equation 2.21 analytically by taking the matrix logarithm of both sides and decomposing the resulting Hamiltonian into a linear combination of Pauli matrices.

$$U = e^{-i\mathscr{H}} = G \tag{2.21}$$

Unfortunately, this often leads to three-body interaction terms which are uncommonly found in nature, and therefore undesirable. It is much more preferable to be able to construct $\mathscr{H}$ out of simpler two-body local spin-interactions. To do this, the Hilbert space must be extended. This can be done by introducing extra qubits to our system called ancillae qubits. This gives a larger degree of freedom to implement a gate with simpler interactions. Daniel Nagaj found that the number of ancillae qubits required to decompose a quantum circuit into a 2-local Hamiltonian using a series of CNOT gates is NL where N are the number of qubits in the gate and L are the number of CNOT gates in your circuit [11]. Since there are three qubits in the construction of a quantum half-adder and it requires a decomposition of six CNOT gates, the number of ancillae required for this method is 18. This is quite a lot of qubits and would be a difficult system to build and maintain. Conversely, the quan-

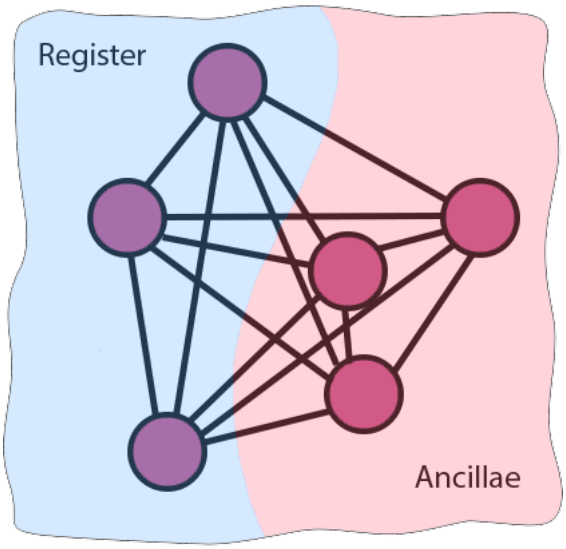


Figure 2.7: A directed graph representing a quantum network split into register and ancillae qubits.

tum gate learning method of implementing a gate can allow for the construction of 2-local Hamiltonians with far fewer ancillae qubits, as shown with the Toffoli gate by L. Banchi et al. [4]. Consider a network of qubits consisting of two composite subspaces: register qubits, R and ancillae qubits, A, as represented in Figure 2.7. Let the initial quantum systems of Q and A be denoted in state vector formalism as $|\psi_Q\rangle$ and $|\psi_A\rangle$, respectively. Note that the time evolution of the overall network is not equal to the tensor product of the individual evolution of both subsystems. That is,

$$U(t)(|\psi_Q\rangle \otimes |\psi_A\rangle) \neq U_Q(t)\,|\psi_Q\rangle \otimes U_A(t)\,|\psi_A\rangle$$

This is because qubits in Q and A interact with each other, and as such are not independent subsystems. Let us redefine the systems in density operator formalism.

$$|\psi\rangle\langle\psi| = |\psi_Q\rangle\langle\psi_Q| \otimes |\psi_A\rangle\langle\psi_A| = \rho_Q \otimes \rho_A = \rho$$

where ρ denotes the state of the overall network, ρ_Q and ρ_A denote the state of subsystems Q and A, respectively.

Using the time-evolution governed by Equation 2.20, a quantum channel, $\mathscr{E}_{J,h}$,

that evolves the state ρ_Q can be generated. This is done by first evolving the overall system ρ, then taking the partial trace over A.

$$\begin{aligned}\mathscr{E}_{J,h}[\rho_Q] &= \mathrm{Tr}_A(U(1)\rho U^\dagger(1)) \\ &= \mathrm{Tr}_A(e^{-i\hat{H}}\rho e^{i\hat{H}}) \\ &= \mathrm{Tr}_A(e^{-i\hat{H}}\rho_Q \otimes \rho_A e^{i\hat{H}})\end{aligned} \tag{2.22}$$

Equation 2.22 represents the time evolution of the subsystem Q. It is this operation that ultimately will operate like quantum gate.

Introducing an ideal quantum gate, G, a similarity measure needs to be defined that will allow the direct comparison between the operation of the quantum channel and this ideal quantum gate. For this, the fidelity function shown in Equation 2.13 for the operators acting on the register qubits, $|\psi_Q\rangle$.

$$F(G|\psi_Q\rangle, \mathscr{E}_{J,h}[\rho_Q]) = \langle\psi_Q| G^\dagger \mathscr{E}_{J,h}[\rho_Q] G|\psi_Q\rangle \tag{2.23}$$

By expanding Equation 2.23 out by substituting in Equation 2.22, the likelihood function between the quantum channel, $\mathscr{E}_{J,h}$, and an arbitrary ideal gate, G, has been successfully derived.

$$F(G|\psi_Q\rangle, \mathscr{E}_{J,h}[\rho_Q]) = \langle\psi_Q| G^\dagger \mathrm{Tr}_A(e^{-i\hat{H}}\rho_Q \otimes \rho_A e^{i\hat{H}}) G|\psi_Q\rangle \tag{2.24}$$

It would be easy to stop here and take Equation 2.24 to be our objective function to maximise to 1. A problem with this is that the likelihood function is calculated over a random sampling of $|\psi_Q\rangle$, this makes it difficult to ensure maximisation for all states $|\psi_Q\rangle$. A way around this is to calculate the average gate fidelity $\bar{F}$, shown

in Equation 2.15.

$$\begin{aligned}\bar{F}(G|\psi_Q\rangle, \mathscr{E}_{J,h}[\rho_Q]) &= \int_{|\psi_Q\rangle} F(G|\psi_Q\rangle, \mathscr{E}_{J,h}[\rho_Q]) d\psi_Q \\ &= \frac{1}{D+1} + \frac{1}{D(D+1)} \sum_{i,j,k,l} G^*_{ik} \mathscr{E}^{ij,kl}_{J,h} G_{jl} \\ &= \frac{1}{D+1} + \frac{1}{D(D+1)} \sum_{i,j,k,l} G^*_{ik} \langle q_i| \mathscr{E}_{J,h}[|q_k\rangle \langle q_l|] |q_j\rangle G_{jl} \\ &= \frac{1}{D+1} + \frac{1}{D(D+1)} \sum_{i,j,k,l} G^*_{ik} \langle q_i| \mathrm{Tr}_A(e^{-i\hat{H}} |q_k\rangle \langle q_l| \otimes \rho_A e^{i\hat{H}}) |q_j\rangle G_{jl}\end{aligned}$$

This function for the average gate fidelity can be used for the optimisation – but it would be much more preferable to use the vectorised form, seen in Equation 2.19.

2.4 Differential Evolution

Optimising the average gate fidelity is no easy task. This is due to the high dimensionality and non-convexity of the average gate fidelity function. Optimisation techniques that require minimising gradients are no use to us as the chances of getting stuck in a local optima is high. A class of optimisation algorithms that are suited for dealing with the global optimisation of many parameter functions are known as Evolutionary Algorithms. An example of an EA is Differential Evolution. DE is a robust, stochastic, population-based optimisation algorithm first introduced by R. Storn and K. Price that can converge quickly to find the global optimum of functions that are non-differentiable, non-continuous or have many local optima [12]. Combined with optimisation strategies based on fidelity statistics in [4], DE would be suitable when dealing with larger quantum networks.

2.4.1 Initialisation

To start off the differential evolution algorithm, the search-space bounds for each parameter must be defined. That is, for every parameter in the objective function set an upper and lower bound.

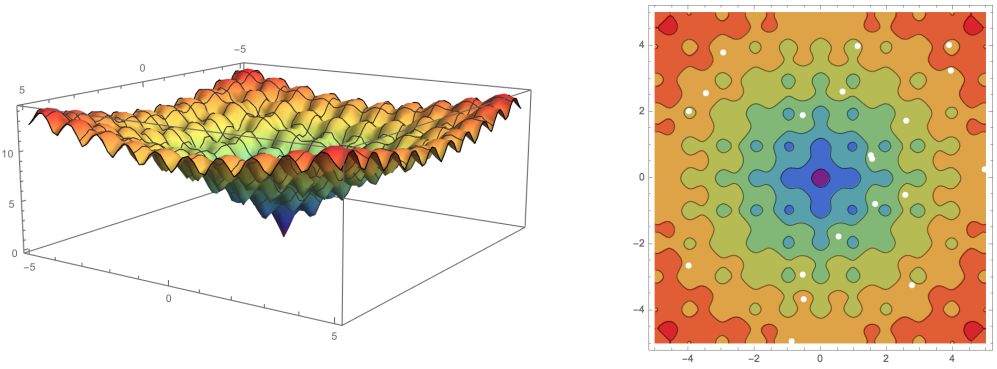
$$\lambda_j^L \leq \lambda_j \leq \lambda_j^U$$

By randomly selecting parameters uniformly on their bounded intervals, a D-dimensional vector of parameters, known as an agent, can be constructed.

$$\mathbf{x} = [\lambda_1, \lambda_2, ..., \lambda_D]$$

The search-space is then populated with N initial agents, labelled as first generation candidates. An example of this instantiate can be seen in Figure 2.8(b).

$$\mathbf{X}_1 = [\mathbf{x}_{1,1}, \mathbf{x}_{2,1}, ..., \mathbf{x}_{N,1}]$$



((a)) ((b))

Figure 2.8: (a) 3D representation of Ackley's function. (b) Twenty agents instantiated on the contour plot of Ackley's function.

2.4.2 Mutation

For the next step, each of the N agents to undergo mutation. Mutation is an important step in the process of DE and determines the size and direction in which the individual agents are perturbed, this allows the agents to traverse the search-space in a stochastic manner. A mutant vector is generated according to,

$$\mathbf{v}_{i,G+1} = \mathbf{x}_{r_1,G} + F(\mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G}) \tag{2.25}$$

$$r_1, r_2, r_3 \in \{1, 2, ..., N\} \qquad F \in [0, 2]$$

where $\mathbf{x}_{r_1,G}, \mathbf{x}_{r_2,G}, \mathbf{x}_{r_3,G}$ are random, unique, agents within the population, G is the generation number, and F is the control parameter responsible for the magnitude of perturbation.

2.4.3 Recombination

Recombination is introduced to increase the diversity of the perturbed parameter vectors. The trial vector $u_{i,G+1}$ is constructed from elements of $x_{i,G}$ and $v_{i,G+1}$

$$u_{j,i,G+1} = \begin{cases} v_{j,i,G+1} & \text{rand}_{j,i} < CR \\ x_{j,i,G} & \text{otherwise} \end{cases} \qquad \text{rand}_{j,i} \sim U[0,1] \quad CR \in \{0,1\} \tag{2.26}$$

where CR is another control parameter known as the crossover rate. The amount of diversity introduced to the system is dependent on CR.

2.4.4 Selection

The target vector $x_{i,G}$ and trial vector $v_{i,G+1}$ are now compared. The vector with the lowest objective function value is selected for the next generation.

$$x_{i,G+1} = \begin{cases} u_{i,G+1} & \text{if } \ f(u_{i,G+1}) < f(x_{i,G}) \\ x_{i,G} & \text{otherwise} \end{cases} \tag{2.27}$$

where $f(x_{i,G})$ is the objective function, in our case, the average gate fidelity. Mutation, recombination, and selection continue until some stopping criterion is met.

2.4.5 Self-Adaptive Differential Evolution

Differential Evolution is incredibly sensitive to three control-parameter: the number of agents N, the mutation factor F, and the crossover rate CR. These control parameters determine the rate of convergence of our optimisation, and whether or not it will find the global optimum or get stuck in a local optima. A way to choose these parameters is by trial-and-error by running a few tests with different parameters and see how the optimisation behaves. Unfortunately, this isn't pos-

sible with expensive functions such as the average gate fidelity for the obvious reasons of taking too long to run a single optimisation. An alternative approach is to vary both F and CR after each generation, G. This is known as Self-Adaptive DE.

After each generation, the control parameters F and CR are updated as follows:

$$F_{i,G+1} = \begin{cases} F_l + r_1 F_u & \text{if} \quad r_2 < \kappa_1 \\ F_{i,G} & \text{otherwise} \end{cases} \tag{2.28}$$

$$CR_{i,G+1} = \begin{cases} r_3 & \text{if} \quad r_4 < \kappa_2 \\ CR_{i,G} & \text{otherwise} \end{cases} \tag{2.29}$$

where F_l, F_u are the lower and upper bounds of the mutation parameter respectively. The quantities r_j with $j \in [1,2,3,4]$ are uniformly distributed random variables such that $r_j \sim U[0,1]$, and κ_1, κ_2 are the probabilities to adjust the control parameters.

This has been found to be effective in decreasing the likelihood of getting stuck in a local optima and allows for higher-fidelity optimisations [13]. Specifically when $\kappa_1 = 0.1$, $\kappa_2 = 0.1$, $F_u = 0.9$ and, $F_l = 0.1$.

Chapter 3

Method

This section of the thesis will outside the methods and ideas behind constructing a quantum gate learning scheme. Firstly, it talks a bit about QuTiP, the package used to handle the quantum mechanical framework. The second section covers how to construct a quantum half-adder matrix using QuTiP. The third section shows how to construct a qubit network Hamiltonian with specific constraints and how to find the quantum channel for the register qubits. The fourth sections demonstrates how I implemented the vectorised average gate fidelity between the quantum channel and the ideal gate. The last section goes into detail on how I programmed a parallel version of self-adaptive differential evolution to optimise the average gate fidelity. The code snippets I show in this section should be enough to understand how everything fits together, but to get the full-picture the entirety of my code can be found in a GitHub repository referenced in Appendix A.

3.1 QuTiP

In order to simulate any quantum behaviour you need a way to work with the quantum mechanical framework of physics. Python, being an open-source programming language, has an extremely extensive list of tools and packages to use. Amongst this is QuTiP (Quantum Toolbox in Python), a useful package for simulating the dynamics of quantum systems. QuTiP allows you to define quantum mechanical objects such as bra-kets, operators and superoperators. I am particularly interested in its ability to perform tensor products, partial traces, and the matrix exponential.

QuTiP will be largely used for encoding a quantum half-adder within the unmodulated dynamics of a qubit network.

3.2 Designing the ideal Quantum Half-Adder

To implement a quantum gate within a qubit network, we need to create an object that represents the ideal quantum half-adder. Referring back to Figure 2.4, constructing the quantum half-adder matrix can be done by breaking the gate down into two simpler gates: Toffoli and CNOT. QuTiP, handily, has predefined functions that return Toffoli and CNOT matrices. Therefore, the quantum half-adder can be defined as the following:

```
def quantum_half_adder():
    return tensor(cnot(), qeye(2)) * toffoli()
```

This operation returns an 8x8 matrix, exactly as shown in Equation 2.7. To ensure that this object truly represents the quantum half-adder, we can test its transformation on the three-qubit binary states.

```
for i in range(2):
    for j in range(2):
            start_psi0 = tensor(basis(2, i), basis(2, j), basis(2, 0))
            end_psi0 = quantum_half_adder() * start_psi0
```

The states transform exactly how we would expect and yields the truth table seen in Figure 2.5.

3.3 Construction of a Qubit Network

The quantum half-adder requires three qubits to operate on. To encode the gate into the unmodulated dynamics of a qubit network, it requires more than three qubits as

three-body interactions emerge in the analytical solution, shown in Equation 3.1.

$$\begin{aligned}\mathscr{H} =& \frac{\pi}{8}(\hat{I}^{\otimes 3} - \sigma_1^Z - \sigma_2^X - \sigma_2^Y + \sigma_3^X - \sigma_2^X \otimes \sigma_3^X \\ &+ \sigma_2^X \otimes \sigma_1^Z - \sigma_3^X \otimes \sigma_1^Z + \sigma_2^Y \otimes \sigma_1^Z \\ &- \sigma_2^X \otimes \sigma_3^X \otimes \sigma_1^Z - \sigma_3^X \otimes \sigma_2^Y \otimes \sigma_1^Z)\end{aligned} \tag{3.1}$$

As an initial starting point to see whether we can constrain the amount of qubits needed, I start with a network of four qubits; three register, and one ancillae. If we assume all combinations of pairwise spin-interactions and a local magnetic field in every direction around each qubit, the Hamiltonian will consist of 48 parameters. The dimensionality of this problem is considerably large, therefore I decided to remove some of the interactions the qubit could experience. Cross-term spin pairwise-interactions are undesirable due to being so uncommon in nature. This simplifies the problem to only three interactions, XX, YY, and ZZ. A visualisation of this can be seen in Figure 3.1, note that the local magnetic fields are not represented in this diagram.

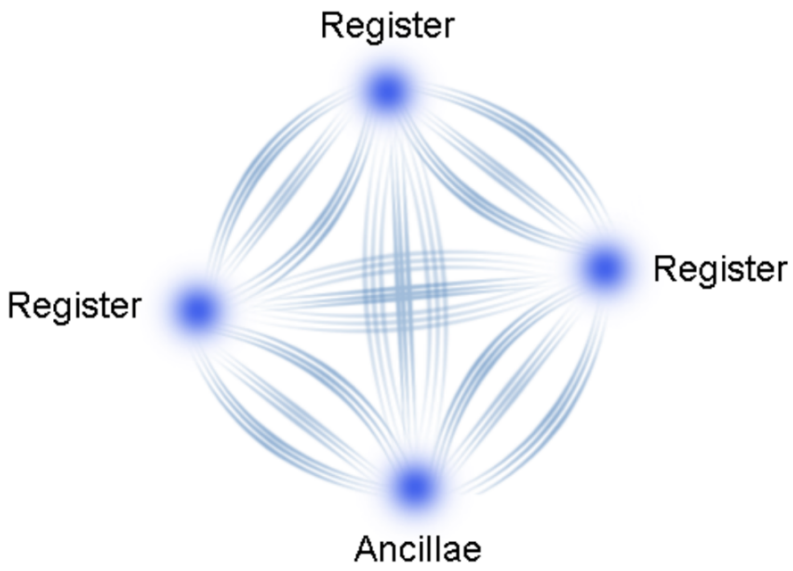


Figure 3.1: A network of four qubits interacting with each other via XX, YY, and ZZ spin pairwise-interactions.

The Hamiltonian of our four qubit system can be extracted from the generalised form in Equation 2.20.

$$\begin{aligned}\mathscr{H} = &\frac{1}{4}(J_{12}^{XX}\sigma_1^X \otimes \sigma_2^X + J_{12}^{YY}\sigma_1^Y \otimes \sigma_2^Y + J_{12}^{ZZ}\sigma_1^Z \otimes \sigma_2^Z \\ &+ J_{13}^{XX}\sigma_1^X \otimes \sigma_3^X + J_{13}^{YY}\sigma_1^Y \otimes \sigma_3^Y + J_{13}^{ZZ}\sigma_1^Z \otimes \sigma_3^Z \\ &+ J_{14}^{XX}\sigma_1^X \otimes \sigma_4^X + J_{14}^{YY}\sigma_1^Y \otimes \sigma_4^Y + J_{14}^{ZZ}\sigma_1^Z \otimes \sigma_4^Z \\ &+ J_{23}^{XX}\sigma_2^X \otimes \sigma_3^X + J_{23}^{YY}\sigma_2^Y \otimes \sigma_3^Y + J_{23}^{ZZ}\sigma_2^Z \otimes \sigma_3^Z \\ &+ J_{34}^{XX}\sigma_3^X \otimes \sigma_4^X + J_{34}^{YY}\sigma_3^Y \otimes \sigma_4^Y + J_{34}^{ZZ}\sigma_3^Z \otimes \sigma_4^Z) \\ &+ \frac{1}{2}(h_1^X\sigma_1^X + h_1^Y\sigma_1^Y + h_1^Z\sigma_1^Z \\ &+ h_2^X\sigma_2^X + h_2^Y\sigma_2^Y + h_2^Z\sigma_2^Z \\ &+ h_3^X\sigma_3^X + h_3^Y\sigma_3^Y + h_3^Z\sigma_3^Z \\ &+ h_4^X\sigma_4^X + h_4^Y\sigma_4^Y + h_4^Z\sigma_4^Z)\end{aligned} \tag{3.2}$$

Noting that this system spans a four-dimension Hilbert space $H^{\otimes 4}$, the expression on the RHS of Equation 3.2 can be constructed by computing the individual two-dimensional Hilbert space contributions of each pairwise-spin as well as the local magnetic field of each qubit, spanning them into the higher four-dimensional Hilbert space, and summing them into the network Hamiltonian.

```
    '''
    Calculates the total hamiltonian of the network.
    '''
    def hamiltonian(self, params, constraints):
        # Interaction strength and local magnetic field strength parameters.
        J = params[:self.number_of_interaction_params()]
        h = params[self.number_of_interaction_params():self.number_of_params()]
        H = 0

        # Hamiltonian contribution from spin-interactions.
        j_iter = 0
        for interacting_qubits in self.get_interactions():
```

```
            H_contribution = 0
            for interaction_type in self.interaction_types:
                operator_chain = [qeye(2)] * self.get_number_qubits()
                operator_chain[interacting_qubits[0]] = interaction_type[0]
                operator_chain[interacting_qubits[1]] = interaction_type[1]

                H_contribution += J[j_iter] * tensor(operator_chain)
                j_iter += 1

            H += 1./4 * H_contribution

        # Hamiltonian contribution from local magnetic fields.
        h_iter = 0
        for qubit_number in range(self.get_number_qubits()):
            H_contribution = 0
            for interaction_type in spin_matrices:
                operator_chain = [qeye(2)] * self.get_number_qubits()
                operator_chain[qubit_number] = interaction_type
                H_contribution += h[h_iter] * tensor(operator_chain)
                h_iter += 1

            H += 1./2 * H_contribution

        return H
```

As a result of spanning over a four-dimension Hilbert space, the form of the Hamiltonian will be a 16x16 matrix.

Although the Hamiltonian plays a very important role in implementing a quantum gate, what we are really interested in is the time-evolution of the register qubit subsystem, given by the quantum channel in Equation 2.22. QuTiP makes it extremely easy to compute the quantum channel as it supplies us with a matrix

exponential and partial trace function.

```
def quantum_channel(H, quantum_state):
    U = (-complex(0, 1) * H).expm()
    E = (U * quantum_state * U.dag()).ptrace(4)
    return E
```

The resulting QuTiP object is an 8x8 matrix representing the time-evolution of the register qubits. The dimensionality of this object is perfect, as it matches the dimensionality of the quantum half-adder matrix. Now we know how to compute these two objects, the quantum channel and the quantum half-adder, we need to compute the average gate fidelity between them.

3.4 Implementing Average Gate Fidelity

The average gate fidelity, as discussed in Chapter 2, can be vectorised. It is this vectorised version that I am interested in implementing, as the original definition requires four for loops to sum over each of the indices, which is highly inefficient. The first step is to create the $|\Xi\rangle$ object found in Equation 2.17. This is first done by defining the computational basis $|i\rangle$ in $H^{\otimes 3}$ and the ancillae state $|\phi_A\rangle$.

```
def basis_8(i):
    a = basis(8, i)
    a.dims = [[2, 2, 2], [1, 1, 1]]
    return a

def xi():
    xi = 0
    for i in range(8):
        xi += 1./8**0.5 * tensor(basis_8(i), basis_8(i), ancillae_state)
    return xi
```

As $|\Xi\rangle$ is a vector defined in $H^{\otimes 7}$, in order to act the time evolution of the system, U, upon it, we must span the dimensionality of U from $H^{\otimes 4}$ to $H^{\otimes 7}$. To do this, the

tensor product between the identity in $H^{\otimes 3}$ and U is performed.

$$U_{4,5,6,7} = I^{\otimes 3} \otimes U$$

Then it is possible to construct the Choi-matrix $\rho_{\mathscr{E}}$, seen in Equation 2.18.

```
def choi_matrix(U):
    U_4567 = tensor(e, U) * xi()
    return (U_4567 * xi() * xi().dag() * U_4567.dag()).ptrace(0, 1, 2, 3, 4, 5)
```

The last thing to compute before the vectorised average gate fidelity can be fully constructed is the vectorised gate operation $|G\rangle$. This is done by going through each element of the quantum half-adder matrix and mapping it onto its corresponding basis state.

```
def vectorised_gate():
    gate_vec = 0
    for i in range(8):
        for j in range(8):
            gate_vec += ideal_gate.data[i, j] * tensor(basis_8(j), basis_8(i))
    return gate_vec
```

The average gate fidelity as shown in Equation 2.19 can now be created. Putting all these elements together, the code becomes.

```
def average_gate_fidelity(params):
    H = network.hamiltonian(params)
    U = (-j * H).expm()
    D = ideal_gate.shape[0]

    overlap = vectorised_gate.dag() * choi_matrix(U) * vectorised_gate()
    return -(1./(D + 1) + 1./(D + 1) * abs(overlap[0][0][0]))
```

Note the negative sign when returning the average gate fidelity. This will become useful when using differential evolution, as this algorithm tries to find the *minimum*

of a function.

The vectorised approach to calculating the average gate fidelity had a significant speed up over the hard-coded for loop version. The time taken to perform this calculation went from ~ 1.2s to ~ 0.2s.

3.5 Implementing Differential Evolution

Python has an extensive library for optimisation, known as *SciPy*. Amongst the numerical optimisation tools they provide is Differential Evolution. When working with high-dimensional, non-convex functions, such as the average gate fidelity, differential evolution requires a large amount of initial agents to provide a chance of finding the global minimum, but becomes slow in the process. Unfortunately, SciPy's differential evolution doesn't allow for parallelisation across a cluster of computers and is incapable of optimising the average gate fidelity due to its high-dimensionality and computation time. Therefore, I had to program my own version of differential evolution that can run in parallel over a cluster of computers.
Differential evolution first requires the instantiation of agents randomly chosen across a bounded search-space.

```
# Generates an initial population of agents, given bounds.
def generate_agents(N, number_of_parameters, bounds):
    agents = []
    for i in range(N):
        agent = []
        for j in range(number_of_parameters):
            agent.append(uniform(bounds[j][0], bounds[j][1]) )

        agents.append(agent)
    return agents
```

Since this is self-adaptive differential evolution, every time a generation of agents go through the procedure of mutation and recombination, the control parameters F

and *CR* must change abiding to the criteria seen in Equation 2.28 and 2.29.

```
# Adaptive Parameters
    r1 = uniform(0, 1)
    r2 = uniform(0, 1)
    r3 = uniform(0, 1)
    r4 = uniform(0, 1)
    mu_l = 0.1
    mu_u = 0.9
    kappa1 = 0.1
    kappa2 = 0.1
    if (r2 < kappa1):
        F = mu_l + r1 * mu_u
    if (r4 < kappa2):
        CR = r3
```

Differential evolution then goes to the mutation and recombination step, where for each agent in the population, three random points from the population must be chosen. A random number must also be chosen to decide whether mutation is carried out by Equation 2.25 or not. This results in a trial vector.

```
# Finds three unique random indexes in an array.
def three_agents(x_index, population):
    index1 = x_index
    index2 = x_index
    index3 = x_index

    while(x_index == index1):
        index1 = randrange(0, len(population))

    while(x_index == index2 or index1 == index2):
        index2 = randrange(0, len(population))

    while(x_index == index3 or index1 == index3 or index2 == index3):
```

```
        index3 = randrange(0, len(population))

    return index1, index2, index3
```

```
x = agents[j]
a_index, b_index, c_index = three_agents(j, agents)

a = agents[a_index]
b = agents[b_index]
c = agents[c_index]

R = randrange(0, number_of_parameters)
# Mutation and Recombination
y = []
for k in range(number_of_parameters):
    r_k = uniform(0, 1)
    if (r_k < CR or k == R):
        y.append(a[k] + F * (b[k] - c[k]))
    else:
        y.append(x[k])
```

If any parameter within an agent lies outside the set boundaries, they are set to a random number within the boundaries.

```
# Bound parameters within a region.
for y_i in range(len(y)):
    if (y[y_i] > hard_bounds[y_i][1]):
        y[y_i] = uniform(hard_bounds[y_i][0], hard_bounds[y_i][1])
    elif (y[y_i] < hard_bounds[y_i][0]):
        y[y_i] = uniform(hard_bounds[y_i][0], hard_bounds[y_i][1])
```

The final step of differential evolution is selection. This selects the better fidelity between the trial agent and the original agent.

```
def selection ( original , trial ):
    fid_original = average_gate_fidelity ( original )
    fid_trial = average_gate_fidelity ( trial )

    if ( fid_trial < fid_original ):
        return fid_trial
    else :
        return fid_original
```

This process is run over every single point and this makes up a differential evolution iteration.

The process of mutation, recombination, and selection on each agent are independent from one-and-other. This is a key requirement for parallelisation since their operations won't be able to interfere with each other. For parallelisation, I used a communications protocal called Message Passing Interface (MPI). For each iteration of differential evolution, the root node distributes each of the agents to different nodes to perform mutation, recombination, and selection. After they have completed their task, their resulting agent vectors are sent from the nodes back to the root node. This process is repeated until some converge criteria.

A way to implement MPI with Python is to use a package called MPI4Py. MPI procedures such as scattering data across nodes (.scatter()), gathering data from nodes to the main node (.gather()), and waiting for all nodes to be finished with a task before proceeding (.Barrier()) can be done with MPI4Py.

The following piece of code allows us to distribute our agents across multiple nodes.

```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
```

```
size = comm.Get_size()

if (rank == 0):
        agents = generate_agents (number_of_agents, number_of_parameters, bounds)
        agents_array = [agents for i in range(size)]
    else:
        agents_array = []

    agents = comm.scatter(agents_array, root=0)
```

The nodes then run a round of differential evolution and send their resulting agent back to the root node.

```
while(iters < iterations):
    data = agent_procedure(func, agents, number_of_parameters, hard_bounds,
        rank)
    comm.Barrier()
    data = comm.gather(data, root=0)

    if (rank == 0):
        agents = data
        iters += 1
        agents_array = [agents for i in range(size)]

    else:
        agents_array = []

    comm.Barrier()
```

In order to keep track of the current progress of differential evolution, the most optimal point along with its function evaluation are written to a file.

```
# Print file every 50 iterations.
iters = comm.bcast(iters, root=0)
```

```
if ( iters  % 50 == 0):
    if (rank  == 0):
        data  = np. array_split  (data ,  size )
    else :
        data  = []
    agents_to_compare  = comm.scatter(data ,  root=0)
    f,  params = func(agents_to_compare [0])
    evaluated_agents  = comm.gather((f,  params),  root=0)

    if (rank  == 0):
        minimum, optimised_agent = min( evaluated_agents ,  key = lambda t:  t [0])
         write_to_file  ( file_name ,  iters ,  minimum, optimised_agent,  end  -  start )

agents  = comm.scatter( agents_array ,  root=0)
comm.Barrier()
```

Chapter 4

Results

4.1 Quantum Half-Adder in a four qubit network

To set up a qubit network, I first had to think about how I should bound the parameters for different evolution and how many agents I was going to instantiate different evolution with. Since I want to make sure the J, h parameters in the Hamiltonian are physical, I set the boundaries of differential evolution to $[-20, 20]$ for each parameter. These values roughly reflect those seen when optimising the Toffoli gate in a similar network [4]. M. E. H. Pedersen recommended with high-dimensional functions, that in the caliber of 30-dimensions, should be instantiated with around 75 agents [14]. Unfortunately, the average gate fidelity was prone to falling in local minima around this number of agents. Increasing the number of agents to 156 gave a nice result, but was very computationally expensive, requiring three days over a cluster of 32 computers to optimise. Ultimately, the optimisation curve seen in Figure 4.1 was produced. The optimisation of the average gate fidelity rises slowly until around 35000 iterations. Then there is a large increase in the gradient. This is because, at this point, most of the agents in the population are surrounding a optima, which they start travelling down until convergence. This run of differential evolution achieved a high-fidelity of 0.9792 for our qubit network implementing a quantum half-adder. Interestingly, a lot of the parameters for the Hamiltonian that

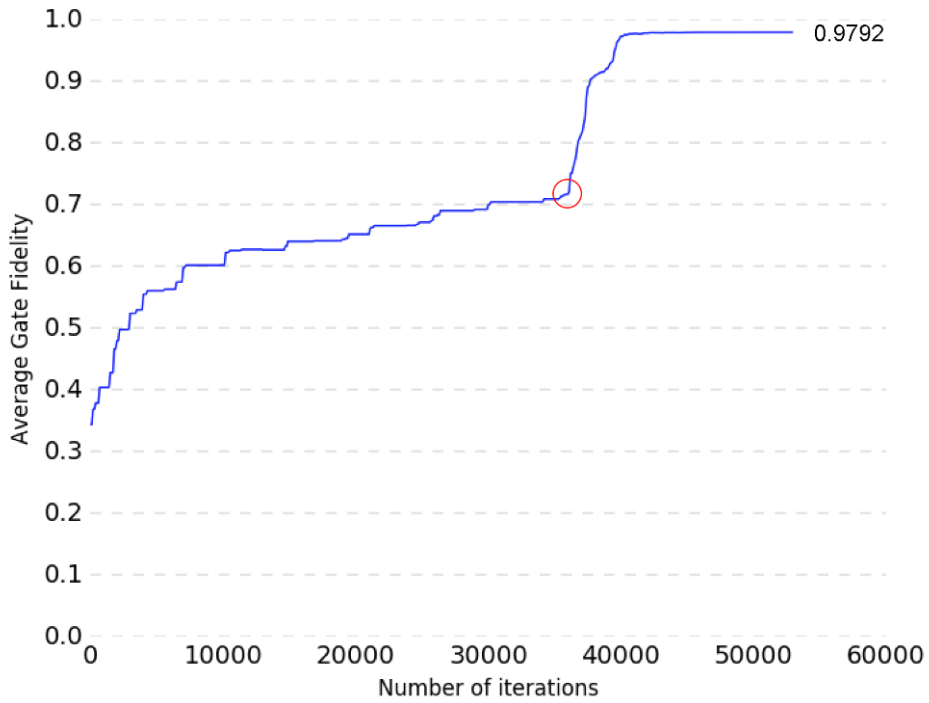


Figure 4.1: The optimisation curve for the average gate fidelity using differential evolution. The red circle denotes where there was a big spike in fidelity.

lead to this result are close to zero.

$$
\begin{array}{lll}
J_{12}^{XX} = 0.0124194 & J_{12}^{YY} = 0.0330664 & J_{12}^{ZZ} = 6.9977747 \\
J_{13}^{XX} = 0.0121806 & J_{13}^{YY} = -0.0272213 & J_{13}^{ZZ} = -17.4561806 \\
J_{14}^{XX} = -0.0077693 & J_{14}^{YY} = -0.0020559 & J_{14}^{ZZ} = -0.9188112 \\
J_{23}^{XX} = -9.5368501 & J_{23}^{YY} = -11.1629320 & J_{23}^{ZZ} = 6.9447932 \\
J_{24}^{XX} = 0.1043263 & J_{24}^{YY} = 0.0139356 & J_{24}^{ZZ} = 0.0692741 \\
J_{34}^{XX} = -17.9716313 & J_{34}^{YY} = -0.1633773 & J_{34}^{ZZ} = 0.2029261 \\
h_1^X = 0.0131634 & h_1^Y = 0.0084270 & h_1^Z = -14.9351325 \\
h_2^X = -4.1712092 & h_2^Y = -2.2729959 & h_2^Z = 3.5019331 \\
h_3^X = -19.8357547 & h_3^Y = 4.7549900 & h_3^Z = -8.6666104 \\
h_4^X = -18.8256969 & h_4^Y = 0.0239434 & h_4^Z = 0.3716292
\end{array}
$$

Motivated by this observation, I decided to iteratively set the parameters with the smallest absolute value to zero and optimise again using Scipy's 'minimize' function which implements the Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimisation method to see if any of them can be discarded completely, as shown in Figure 4.2.

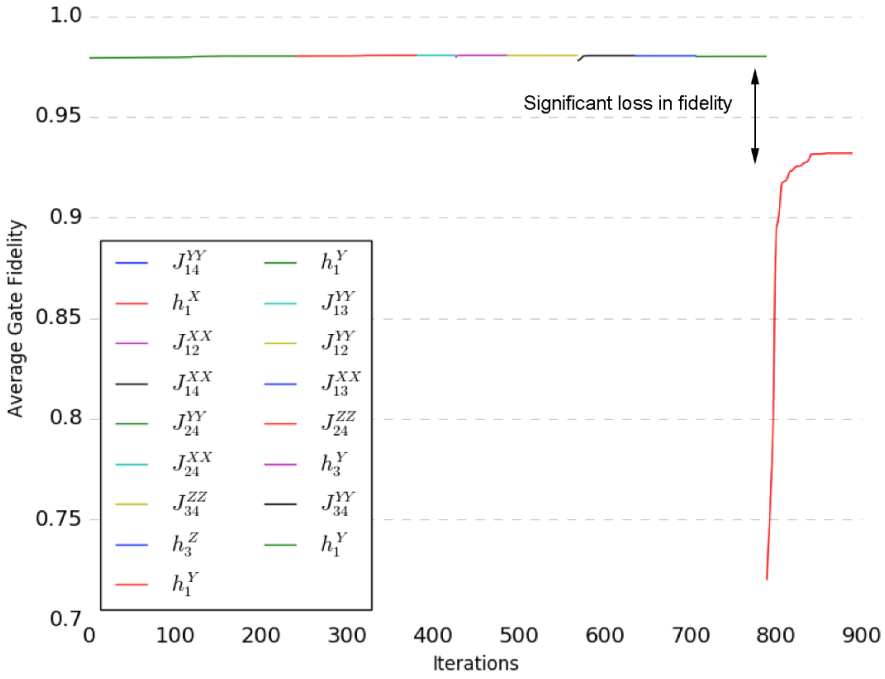


Figure 4.2: Optimisation curves for iteratively setting the smallest parameters to zero and optimising until a significant loss of fidelity occurs.

This iterative minimisation process lasted for 15 iterations until on the 16th, there was a significant loss in the operation of the implemented quantum half-adder. This result means that we can discard 16 of our parameters and still have a fully functioning quantum half-adder with a fidelity of around 0.98. The remaining parameters

are as follows:

$$J_{12}^{ZZ} = 6.9997011$$
$$J_{13}^{ZZ} = -17.3961947$$
$$J_{14}^{ZZ} = 2.4968609$$
$$J_{23}^{XX} = -9.5115536 \quad J_{23}^{YY} = -11.0603813 \quad J_{23}^{ZZ} = 6.8754149$$
$$J_{34}^{XX} = -13.2808596$$
$$h_1^Z = -14.924147$$
$$h_2^X = -4.1277179 \quad h_2^Y = -2.2985079 \quad h_2^Z = 3.5000632$$
$$h_3^X = -22.2314788 \quad h_3^Y = 4.5632352 \quad h_3^Z = -8.7004208$$
$$h_4^X = -18.7864430$$

The visualisation in Figure 4.3 shows the transformation from the complex network with all the interactions to the simplified network with the remaining parameters.

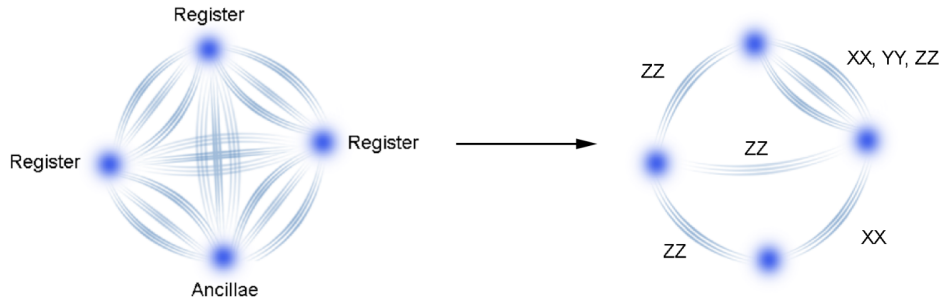


Figure 4.3: Complex network transforms into a simplified network with much fewer interactions after iteratively setting small parameters to 0 and minimising.

It is a point to note that the majority of the network is made up of ZZ interactions, shown to be obtainable in superconducting circuits by Geller et. al [15]. Additionally, McKay et al. showed that it was possible to obtain $(XX+YY)$ interactions in a superconducting circuit [16], lending itself nicely for the interactions between the second and third qubit of this network. This makes the quantum half-adder with this network configuration almost fully implementable in a superconducting circuit.

The only problem is the XX interaction between the third register qubit and the ancillae qubit.

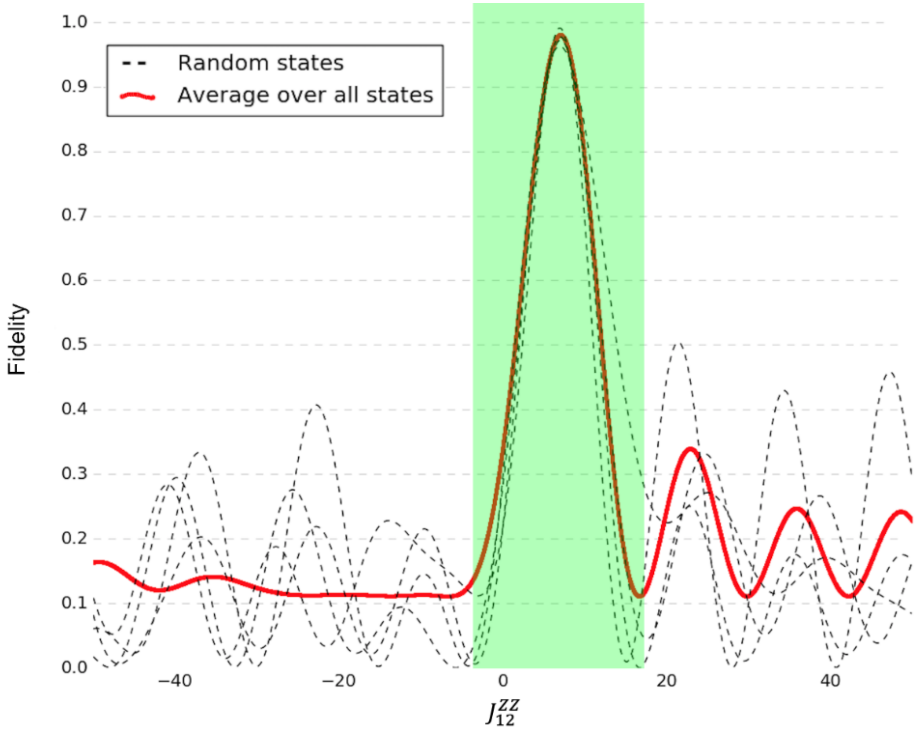


Figure 4.4: The average gate fidelity and fidelity of random states given by the implemented quantum half adder when varying one parameter, J_{12}^{ZZ} around its optimal value.

To fully appreciate the operation of the implemented quantum half-adder, Figure 4.4 displays the average gate fidelity and the fidelity of random states that have been acted on upon by our quantum network whilst varying J_{ZZ}^{12} around its optimal value. It is evident that there only exists one global optima in this dimension, which lies at the optimal value of the J_{12}^{ZZ} parameter, which is a promising result found by differential evolution. Around this optima, the fidelity of the random states are all clustered around the average gate fidelity which is to be expected if our gate is to act like a quantum half-adder for all states. It may be tempting to say that this is the global optima of the average gate fidelity, but this cannot be extended from a one-dimensional representation of the function. Instead, this solution is more likely to be a local optima due to differential evolution having trouble with high-dimensional functions. Many runs of differential evolution on the average gate

fidelity were carried out before an acceptable optima was found.

4.2 Toffoli Gate in a four qubit network

Setting up a network similar to the quantum half-adder; four qubits, parameters bounded by $[-20, 20]$ and instantiating differential evolution with 156 produced the optimisation curve seen in Figure 4.5. This method of optimising a quantum network managed to implement the Toffoli gate with an exceptionally high fidelity of 0.99839. By doing some further optimisation using SciPy's 'minimize' function

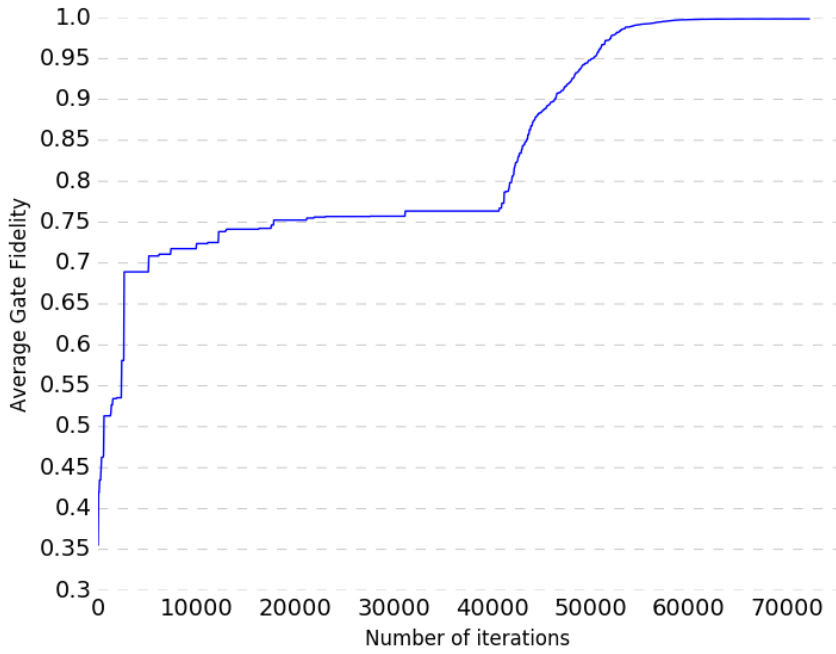


Figure 4.5: Differential evolution optimisation curve for the Toffoli gate. The most optimal average gate fidelity is 0.9984

which uses the BFGS algorithm, the fidelity was able to reach an average gate fidelity of 0.99995, seen in Figure 4.6.

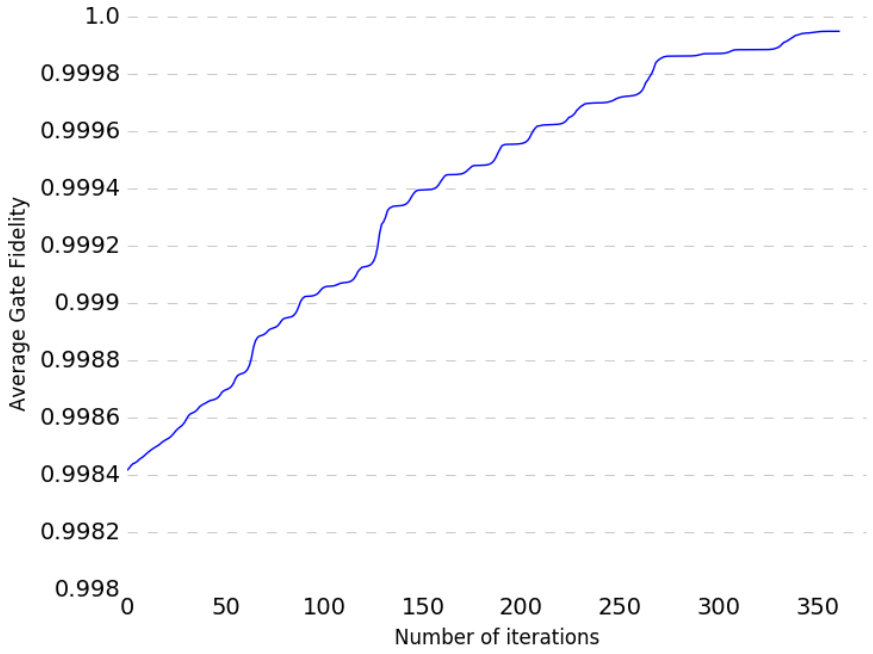


Figure 4.6: Further optimisation using SciPy's BFGS algorithm. The final optimised fidelity is 0.99995

The parameters that produce this fidelity are as follows:

$$
\begin{array}{lll}
J_{12}^{XX} = -0.7565649 & J_{12}^{YY} = -0.4109247 & J_{12}^{ZZ} = 22.2850985 \\
J_{13}^{XX} = 7.910381 & J_{13}^{YY} = 0.4848638 & J_{13}^{ZZ} = 16.8887795 \\
J_{14}^{XX} = -18.6353619 & J_{14}^{YY} = -2.8153867 & J_{14}^{ZZ} = -21.281077 \\
J_{23}^{XX} = 21.7122241 & J_{23}^{YY} = 1.2952030 & J_{23}^{ZZ} = 13.2333797 \\
J_{24}^{XX} = 25.1026156 & J_{24}^{YY} = 3.1606596 & J_{24}^{ZZ} = -0.4798341 \\
J_{34}^{XX} = -16.4636992 & J_{34}^{YY} = 5.3127875 & J_{34}^{ZZ} = -5.4082343 \\
h_1^X = 13.2817502 & h_1^Y = 0.0003616 & h_1^Z = -17.1112761 \\
h_2^X = -1.6577358 & h_2^Y = -0.0014299 & h_2^Z = -17.8507379 \\
h_3^X = -18.0614019 & h_3^Y = 0.0004289 & h_3^Z = 14.506291 \\
h_4^X = -8.3998226 & h_4^Y = 0.0000865 & h_4^Z = -10.9361361
\end{array}
$$

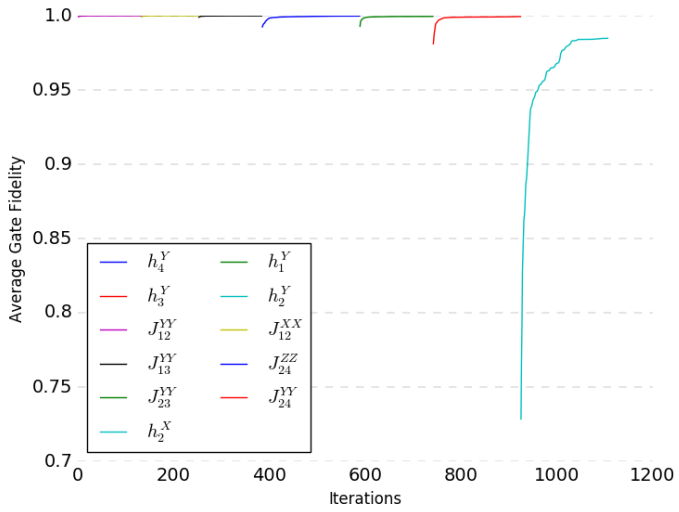


Figure 4.7: Iteratively setting the smallest absolute value to zero and minimising using BGFS.

Similarly to the quantum half-adder, many of the parameters were close to zero. Following the same procedure of iteratively setting the parameters to zero and minimising allows the removal of ten parameters without the fidelity significantly dropping, this can be seen in Figure 4.7. This achieves an average gate fidelity of 0.9994 with the following parameters:

$$J_{12}^{ZZ} = 22.3748801$$

$$J_{13}^{XX} = 15.2389914 \qquad J_{13}^{ZZ} = 9.4953760$$

$$J_{14}^{XX} = -14.6943610 \qquad J_{14}^{YY} = -3.3847032 \qquad J_{14}^{ZZ} = -22.5375930$$

$$J_{23}^{XX} = 17.75359290 \qquad J_{23}^{ZZ} = 15.9653634$$

$$J_{24}^{XX} = 24.182184$$

$$J_{34}^{XX} = -13.4459607 \qquad J_{34}^{YY} = 3.38660686 \qquad J_{34}^{ZZ} = -3.5883792$$

$$h_1^X = 15.5603445 \qquad h_1^Z = 16.1047022$$

$$h_2^X = -3.3492931 \qquad h_2^Z = -18.4540431$$

$$h_3^X = -20.0733950 \qquad h_3^Z = 12.5092083$$

$$h_4^X = -8.5541109 \qquad h_4^Z = -10.9217511$$

Chapter 5

Discussion and Conclusion

To conclude, I found that it is possible to encode a quantum half-adder into the unmodulated dynamics of a four qubit network consisting of three register qubits and one ancillae qubit interacting via pairwise spin-interactions with a high average gate fidelity of 0.98. This was done by using self-adaptive differential evolution which I had to parallelise using the communications protocol, MPI. After optimising, I found that a lot of the parameters of the network could be set to zero–the second register qubit and ancillae qubit can be totally disconnected–without a significant loss of fidelity. An interesting aspect of the parameters for the quantum-half-adder are that many of the non-zero interactions are implementable in a superconducting circuit – showing promise to the development of quantum hardware for a quantum half-adder. Unfortunately, the XX-interaction between the third register and the ancillae hasn't been obtained in a superconducting circuit yet. Conversely to the method applied to remove interactions from the network whilst still preserving the fidelity of the gate, perhaps it is possible to add interactions back into the network without the fidelity suffering. It would be interesting to see whether it is possible to add a YY-interaction between the third register and ancillae qubit. Since $(XX+YY)$ and ZZ interactions are possible in superconducting circuits, this would allow us to fully realise this quantum gate. Although the fidelity of this quantum half-adder is high, a slightly higher fidelity would be needed if it were to be used in academia. It is entirely possible that the solution found is not the global optima, and a better solution is yet to be found. A way a better solution may be found is by instantiating

differential evolution with more agents, this would allow for more search-space to be traversed and a lower the probability of getting stuck in a local optima. The main downfall for this method is that the computation is highly expensive; requiring a cluster of perhaps hundreds of computers. Since it was found that many of the parameters to implement the quantum half-adder were redundant, an idea would be to try and reduce the dimensionality of the problem before using differential evolution to optimise. This would allow for differential evolution to run more efficiently with more initial agents – increasing the chances of finding a global optima.

Additionally, with this method, I found a separate solution to what is already known for encoding the Toffoli gate within a four qubit network with an exceptionally high average gate fidelity of 0.99995. Although the parameters for the Toffoli gate aren't as implementable as the previously found using stochastic gradient descent [4], this gives insight on the flexibility of choosing parameters to implement a quantum gate. This interpretation gives rise to the idea that the solution found for the quantum half-adder is sub-optimal and there could possibly exist many local optima above 0.98.

The next steps, outside of finding a better solution for the quantum half-adder, would be to see if it is possible to put these quantum half-adder networks together in order to produce a quantum full-adder. This would require finding a way to join the interactions between qubit networks together. Half-adders are one of the core components for performing calculations on a computer. The successful implementation of the quantum half-adder within the always-on interactions of a four qubit network is a step towards being able to build fast and fault-tolerant quantum hardware for the core components of a quantum computer.

Appendix A

GitHub - Quantum Gate Learning

All the code necessary to run parallelised self-adaptive differential evolution on a qubit network to encode a generic quantum gate into the unmodulated dynamics of that network can be found on GitHub at `https://github.com/Usefulmaths/MSci-Project`.

Bibliography

[1] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, New York, NY, USA, 10th edition, 2011.

[2] Ehsan Zahedinejad, Joydip Ghosh, and Barry C Sanders. Designing high-fidelity single-shot three-qubit gates: A machine learning approach. *arXiv preprint arXiv:1511.08862*, 2015.

[3] Leonardo Banchi, Nicola Pancotti, and Sougato Bose. Supervised quantum gate" teaching" for quantum hardware design. *arXiv preprint arXiv:1607.06146*, 2016.

[4] Leonardo Banchi, Nicola Pancotti, and Sougato Bose. Quantum gate learning in qubit networks: Toffoli gate without time-dependent control. *npj Quantum Information*, 2:16019, 2016.

[5] Benjamin Schumacher. Quantum coding. *Phys. Rev. A*, 51:2738–2747, Apr 1995.

[6] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Phys. Rev. A*, 52:3457–3467, Nov 1995.

[7] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of*

Computer Science, SFCS '94, pages 124–134, Washington, DC, USA, 1994. IEEE Computer Society.

[8] Arjen K. Lenstra and Jr. Hendrik W. Lenstra, editors. *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1993.

[9] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, 1996. ACM.

[10] A. Uhlmann. The transition probability in the state space of a ł-algebra. *Reports on Mathematical Physics*, 9(2):273 – 279, 1976.

[11] Daniel Nagaj. Universal two-body-hamiltonian quantum computing. *Phys. Rev. A*, 85:032330, Mar 2012.

[12] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[13] Ehsan Zahedinejad, Joydip Ghosh, and Barry C. Sanders. High-fidelity single-shot toffoli gate via quantum control. *Phys. Rev. Lett.*, 114:200502, May 2015.

[14] Magnus Erik Hvass Pedersen. Good parameters for differential evolution. *Magnus Erik Hvass Pedersen*, 2010.

[15] Michael R. Geller, Emmanuel Donate, Yu Chen, Charles Neill, Pedram Roushan, and John M. Martinis. Tunable coupler for superconducting xmon qubits: Perturbative nonlinear model. 2014.

[16] David C. McKay, Stefan Filipp, Antonio Mezzacapo, Easwar Magesan, Jerry M. Chow, and Jay M. Gambetta. A universal gate for fixed-frequency qubits via a tunable bus. 2016.