Genetic Algorithm Optimization of Quantum Gates for an NMR Quantum Information Processor

Amit Devra

A dissertation submitted for the partial fulfilment of BS-MS dual degree in Science

Indian Institute of Science Education and Research Mohali
April 2017

Certificate of Examination

This is to certify that the dissertation titled **Genetic Algorithm Optimization of Quantum Gates for an NMR Quantum Information Processor** submitted by **Amit Devra** (Reg. No. MS12012) for the partial fulfillment of BS-MS dual degree programme of the Institute, has been examined by the thesis committee duly appointed by the Institute. The committee finds the work done by the candidate satisfactory and recommends that the report be accepted.

Dr. Kavita Dorai (Supervisor) | Prof. Arvind | Dr. S. K. Goyal

Dated: April 21, 2017

Declaration

The work presented in this dissertation has been carried out by me under the guidance of Dr. Kavita Dorai at the Indian Institute of Science Education and Research Mohali.

This work has not been submitted in part or in full for a degree, a diploma, or a fellowship to any other university or institute. Whenever contributions of others are involved, every effort is made to indicate this clearly, with due acknowledgement of collaborative research and discussions. This thesis is a bonafide record of original work done by me and all sources listed within have been detailed in the bibliography.

Amit Devra
(Candidate)

Dated: April 21, 2017

In my capacity as the supervisor of the candidates project work, I certify that the above statements by the candidate are true to the best of my knowledge.

Dr. Kavita Dorai
(Supervisor)

Acknowledgment

Firstly, I would like to thank my project guide Dr. Kavita Dorai for her expert guidance. I have been amazingly fortunate to have adviser like her who gave me an opportunity to explore the subject and at the same time giving me the valuable guidance. I must thank them for their patience and encouragement throughout the project.

I am deeply thankful to my committee members Prof. Arvind and Dr. Sandeep Goyal for their insightful comments and suggestions during project work. I am sincerely thankful to IISER, Mohali library.

I am specially thankful to Harpreet Singh for being an amazing mentor and Prithviraj Prabhu for being a part for this project work. Without your guidance and discussions this work would not have been possible.

I am also thankful to the lab and QCQI group members Navdeep, Satnam, Amandeep Singh, Rakesh Sharma, Jyotsana Ojha, Akshay Gaikawad, Jassi, Saxena, Rajendra Bhati for their helpful discussions.

This journey of five years would not have been possible without friends. I would like to thank Himanshu, Rohan, Aditya and specially Priyanka Bansal for making these five years memorable. Thank you for inspiring me throughout the journey and for making these years unforgettable.

I fall short of words to thank my parents and sisters for being very supportive and encouraging throughout my whole life. Their support and unconditional love have helped me and guided through life.

List of Figures

Abstract

The experimental implementation of quantum algorithms on a quantum computer requires the breakdown of unitary operators. Here we considered this as an optimization problem and used genetic algorithms. Genetic algorithms are stochastic search algorithms and a global optimization technique which mimics the behavior of biological evolution in nature. This optimization technique has been widely used for quantum computing applications. We apply this optimization techniques for an NMR quantum information processor and optimized the three-qubit unitary matrices.

The algorithm for an NMR quantum information processor was modified and designed in such a way that the unitary matrices can be implemented using only hard pulses and delays. We mainly focused on three-qubit quantum gates such as Toffoli and Fredkin as they are universal for computation and has much application in various algorithms and protocols. The pulse sequence corresponding to the unitary matrices were time optimal and robust to cope up with the errors associated with the NMR quantum information processing. The optimized pulse sequence for the three-qubit unitary matrices was obtained with very high theoretical fidelity. We experimentally implemented these optimized quantum gates on a system of three coupled NMR qubits and computed the final fidelity.

Contents

Chapter 1

Quantum Computation

"The most important application of quantum computing in the future is the computer simulations of a quantum systems, because that's an application where we know for sure that quantum systems in general cannot be efficiently simulated on a classical computer."

— David Deutsch

1.1 Introduction

The extraordinary progress in the development of the computer technologies usually summarised in the form of Moore's law [Moo65]. Moore's law observes that the the computer powers for a given sum of money doubles approximately every two years. Like any form of exponential growth this doubling soon leads to enormous numbers: computer power increases tenfold every five years, one hundred fold every decade, and so on. It has now held true for almost fifty years and it is tempting to assume that it can continue for many more years [AJ01]. This huge increase in computing power require decrease in the size of electronic components. At these length scales, the classical laws of physics will no longer hold. The world at these length scales has very different properties and follow laws of quantum mechanics. The use of quantum technologies will not only make computers small in size but also fast and efficient. It will allow us to solve some of the problems which are not even possible for a classical computer. In a way quantum computation is the field that investigates the computational power and other properties of computers based on quantum mechanical principles.

1.2 Quantum Bits

A bit is a fundamental unit of information in a classical computer. It can have two possible states: 0 or 1. The analogous concept for quantum computers is a quantum bit or *qubit*. Just like bits, qubits also have a state. The difference between a bit and a qubit is that it can be in a state other than 0 or 1. A qubit can be in a linear combination of 0 and 1, generally known as superposition. The general state of a qubit can be written as,

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

The numbers α and β are complex numbers having normalization condition $|\alpha|^2 + |\beta|^2 = 1$. The states $|0\rangle$ and $|1\rangle$ are known as computational basis which forms an orthonormal basis for this vector space. A qubit can be easily visualized by a Bloch sphere [NC11],

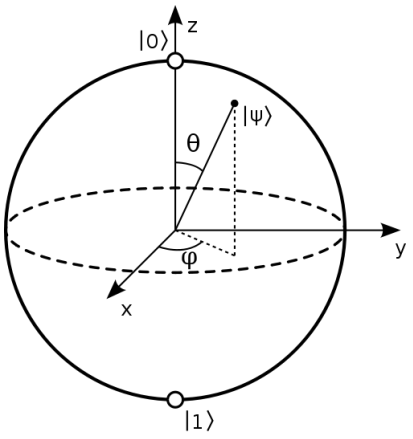


Figure 1.1: Bloch sphere representation of a qubit.

And the most general state as in equation(1.1) can be rewritten as,

$$|\Psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\phi}\sin\frac{\theta}{2}\,|1\rangle \tag{1.2}$$

where the terms θ and ϕ are as shown in the figure 1.1. The general state of a n-qubit system can be written as,

$$|\psi\rangle = \sum_{k=0}^{2^n-1} \alpha_k |k\rangle \tag{1.3}$$

where,

$$\sum_{k=0}^{2^n-1} |\alpha_k|^2 = 1 \tag{1.4}$$

In contrast to classical computation in which the state can be either in 0 or 1, the superposition of states as shown in equation(1.3) means that the state $|\psi\rangle$ is in 2^n states simultaneously. This infers that a quantum computer can out perform a classical computer in terms of speed and efficiency.

1.3 Quantum Gates

Analogous to a classical computer which consist of logic gates to carry information, quantum computers also has these logics known as quantum gates. These quantum gates help to carry information and manipulate them. By combining these quantum gates one can build a quantum circuit which can perform a desired operation. Some properties of quantum gates are-

- Unlike many classical logic gates, quantum gates are reversible.
- Since the evolution of a quantum state is restricted by unitary, so the quantum gates are represented by unitary matrices of order $2^n \times 2^n$, which follows the following property,

$$|\psi(t)\rangle' = U|\psi\rangle \tag{1.5}$$

$$UU^\dagger = U^\dagger U = \mathbb{I} \tag{1.6}$$

Some of the most common quantum gates are given in the following sections.

1.3.1 Single Qubit Gates

- **Pauli-X Gate (NOT Gate)**
  It is equivalent to classical NOT gate. The Pauli-X matrix is given by,

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

  The action of NOT gate is to take state $|0\rangle$ to state $|1\rangle$ and vice versa. Consider the state given in equation(1.1) when written in matrix form will look like,

$$|\Psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

The action of X on above state is,

$$\mathrm{X}\begin{bmatrix}\alpha\\ \beta\end{bmatrix}=\begin{bmatrix}\beta\\ \alpha\end{bmatrix}$$

- **Pauli-Z Gate**

  The Pauli-Z matrix is given by,

$$Z=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$$

  The action of Z gate is to take state $|1\rangle$ to -$|1\rangle$ keeping $|0\rangle$ state unchanged. It will act on $|\Psi\rangle$ as follows,

$$\mathrm{Z}\begin{bmatrix}\alpha\\ \beta\end{bmatrix}=\begin{bmatrix}\alpha\\ -\beta\end{bmatrix}$$

- **Hadamard Gate**

  It is a unique gate which does not have any classical analogue. Its uniqueness takes state from one basis to another basis. It acts on state as,

$$\begin{aligned}\mathsf{H}\,|0\rangle &= \frac{|0\rangle+|1\rangle}{\sqrt{2}}\\ \mathsf{H}\,|1\rangle &= \frac{|0\rangle-|1\rangle}{\sqrt{2}}\end{aligned} \tag{1.7}$$

  The matrix representation of Hadamard(H) gate is given as,

$$H=\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$$

1.3.2 Multiple Qubit Gates

- **Controlled-NOT Gate**

  Controlled-NOT (CNOT) is a two qubit gate from which one of them is control qubit and other one is target qubit. Target qubit is flipped when control qubit is in state $|1\rangle$. Its circuit representation is given by,

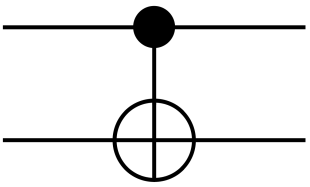


Figure 1.2: Circuit representation of a CNOT gate.

Taking two qubits namely A and B. A is a control qubit and B is target qubit, under the action of CNOT they will vary as,

$$\mathsf{CNOT}\,|A, B\rangle \rightarrow |A, A \oplus B\rangle \tag{1.8}$$

where '$\oplus$' represents addition modulo two function. In terms of the computational basis i.e. $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ the matrix representation is given by,

$$\mathsf{CNOT}= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

CNOT gate when combined with a single qubit quantum gate forms a universal set for quantum computing i.e. any multiple qubit gate may be composed from CNOT and single qubit gates.

- **Toffoli Gate**

  Tofoli is a three qubit gate from which two qubits acts as control and other one acts as a target. Control qubits are unaffected by the action of Toffoli gate and target qubit flips when both the control qubits are in state $|1\rangle$ under the action of Toffoli. Its circuit representation is given by,

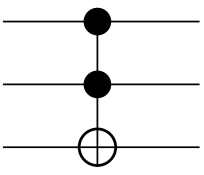


Figure 1.3: Circuit representation of a CNOT gate.

The action will change the qubits as,

$$\mathsf{U}\,|A, B, C\rangle \rightarrow |A, B, C \oplus AB\rangle \tag{1.9}$$

Toffoli gate's matrix representation is given by,

$$\mathsf{U}= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Toffoli gate is a universal for classical computation. It can simulate any irreversible classical logic gate. And it can be easily proved by simulating NAND gate which is a universal gate for classical computation from Toffoli gate.

$$\mathsf{U}\,|A, B, 1\rangle \rightarrow |A, B, 1 \oplus AB\rangle \tag{1.10}$$

The equation 1.10 is simulating NAND gate by considering first two qubits as input and third qubit as output. Third qubit here acts as an ancilla state. So it infers from above that Toffoli gate is a universal for classical computation.

The universality of Toffoli gate ensures that quantum computers are capable of performing any computation which is a classical computer may do. There are many muliple qubit quantum gates with different properties which we will come across in the upcoming chapters. The next chapter will give an introduction of how one can do quantum computation using a physical system.

Chapter 2

NMR Quantum Computing

"On the theoretical side, I was concerned with stochastic resonance."

— Richard Ernst

The basic requirements for experimental quantum computing are well known and are listed by DiVincenzo [DiV00]. In this chapter, we will see how nuclear magnetic resonance (NMR) follows all of these criteria and why is it suitable for quantum computing. Nuclear magnetic resonance differs from other implementations of quantum computer as it uses an ensemble of systems. It uses the spin states of the molecule as a qubit. And the output of an NMR measurement is an average over all the molecule's signal. In the next section, the theory behind nuclear magnetic resonance is explained.

2.1 Basic Concepts on NMR

Resonance is a phenomenon which can be defined as, when a system with natural frequency is excited by an external periodic perturbation of frequency close to that natural frequency then a strong increase in the amplitude of vibration takes place [IO07]. When a particle having magnetic dipole moment is simultaneously placed in a static magnetic field and an oscillating electromagnetic field, resonance occurs. This phenomenon is known as magnetic resonance.

The phenomenon of NMR can be observed for a nuclei having non-vanishing total angular momentum. If the number of protons and neutrons are even then nuclear spin is zero. On the other hand, if there is only an unpaired nucleon then nuclear spin is equal to the total angular momentum of that nucleon. All atomic nuclei having

non-zero nuclear spin posses a magnetic dipole moment (μ).

The Wigner-Eckart theorem gives the direct relation between magnetic dipole moment and nuclear spin as,

$$\mu = \gamma_n \hbar \mathbf{I} \tag{2.1}$$

where γ_n is the gyromagnetic ratio of the nucleus and $\mathbf{I}$ is the total angular momentum. In case of nucleus, total angular momentum commonly refers are spin angular momentum or nuclear spin. Characteristic of nuclear spin are are given by the eigenvalues and the eigen vectors of $\mathbf{I^2}$ and its z-component I_z,

$$\mathbf{I}^2 \left| I, m \right\rangle = I(I+1) \left| I, m \right\rangle \tag{2.2}$$

$$I_z \left| I, m \right\rangle = m \left| I, m \right\rangle \tag{2.3}$$

where I and m are angular momentum quantum number and magnetic quantum numbers respectively. And m can vary from $-I$ to I taking $2I + 1$ values.

The energy involved in NMR experiments is much smaller than the energy spacing between the ground and excited states. One can consider that the nucleus is in ground state permanently i.e fixed I. The energy of a nucleus in such a situation is therefore determined only by magnetic quantum number (m).

2.1.1 Interaction with Static Magnetic Fields

Atomic nuclei with non-zero total angular momentum interact with electromagnetic fields present in their environment through the nuclear magnetic dipole moment. The basic interaction necessary to understand NMR is the Zeeman interaction. It is the interaction between the nuclear magnetic dipole and an external static magnetic field which gives rise to manifold of energy levels for the nucleus depending up on the orientation with respect to the static magnetic field. The absorption and irradiation of energy associated with transitions between these levels constitute the physical phenomena observed in an experiment of magnetic resonance. The figure 2.1 describes how a nucleus having a magnetic moment μ rotates in an external static magnetic field B_0.

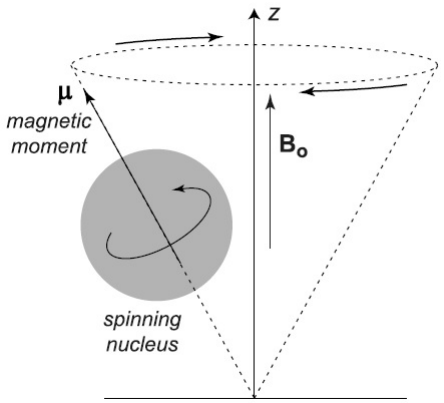


Figure 2.1: Precession of a nucleus in a static magnetic filed.

The classic interaction between a body with magnetic dipole moment μ and an external static magnetic field B_0 is descried by potential energy(U) as,

$$U = -\mu.B_0 \tag{2.4}$$

and have an associated torque (τ),

$$\tau = \mu \times B_0 \tag{2.5}$$

When an atomic nucleus with magnetic dipole moment (μ) is placed in an external static magnetic field B_0, the nuclear states assume different energy values depending on the orientation of the nuclear spin. This splitting is known as nuclear Zeeman effect and the Hamiltonian corresponding to this is known as Zeeman Hamiltonian,

$$\mathcal{H}_{\mathcal{Z}} = -\mu.B_0 = -\mu_z.B_0 = -\gamma_n \hbar B_0 I = -\hbar\omega_L I_z \tag{2.6}$$

where ω_L is the Larmor frequency. Larmor frequency is defined as the frequency with which the angular momentum vector precess about the external magnetic field. The energy eigenvalues corresponding to the Hamiltonian is given by,

$$E_m = -m\hbar\omega_L \tag{2.7}$$

Hence, there are 2I+1 energy levels equally spaced by $\hbar\omega_L$.

2.1.2 Interaction with Radio Frequency Field

In NMR, for nuclear spins the Larmor frequencies are of the order of MHz, so the excitation is achieved by radio frequency (RF) field. This can be understood by considering the effect of a second time-dependent magnetic field $B_1(t)$ which is applied perpendicular to the static magnetic field, say in x direction.

$$B_1(t) = 2B_1 cos(\omega_{rf} t + \phi)\hat{\boldsymbol{i}} \tag{2.8}$$

where ω_{rf} and ϕ are the frequency and the phase respectively. The RF Hamiltonian is give by,

$$\mathcal{H}_{\mathcal{RF}} = -\mu.B_1(t) = -\gamma_n \hbar B_0 I_x[2B_1 cos(\omega_{rf} t + \phi)] \tag{2.9}$$

The result of this is that when the frequency of the RF field is close to the Larmor frequency i.e., on resonance, the transition between the eigenstates are induced.

The semi classical interpretation for the excitation of nuclear is spin is obtained by considering the linearly polarized magnetic field $B_1(t)$ as composed of two circularly polarized fields rotating with same frequency but in opposite directions,

$$B_1(t) = B_1^+(t) + B_1^-(t) \tag{2.10}$$

$$B_1^+(t) = B_1[cos(\omega_{rf} t + \phi)\hat{\boldsymbol{i}} + sin(\omega_{rf} t + \phi)\hat{\boldsymbol{j}}] \tag{2.11}$$

$$B_1^-(t) = B_1[cos(\omega_{rf} t + \phi)\hat{\boldsymbol{i}} - sin(\omega_{rf} t + \phi)\hat{\boldsymbol{j}}] \tag{2.12}$$

In the condition of resonance, the field $B_1^-(t)$ rotates around z-axis, whereas $B_1^+(t)$ rotates in opposite sense. Consider a coordinate system rotating around z-axis with frequency the same as $B_1^-(t)$ such that $B_1^-(t)$ is stationary and $B_1^+(t)$ is rotates with twice the Larmor frequency. Hence, $B_1^-(t)$ will have an effect on the nuclear spins.

If the frequency of RF is ω_{rf} not equal to ω_L, the precession of the magnetic moments in the rotating frame in an effective magnetic field given by,

$$B_{eff} = (B_0 - \frac{\omega_{rf}}{\gamma_n})\hat{\boldsymbol{j}} + B_1\hat{\boldsymbol{i}} \tag{2.13}$$

This effective field will cause the net magnetization to deviate from z-axis. On resonance the magnetization (M) precess in the frame around x'- direction with the nutation frequency given by $\omega_1 = \gamma_n B_1$ and with angle $\theta_p = \gamma_n B_1 t_p$, where t_p is the

time for which RF was on. This transient RF is known as a RF pulse, and t_p is the pulse duration. If $\theta_p = \pi/2$ it is called a is $\pi/2$ pulse and $\theta_p = \pi$ is known as a π

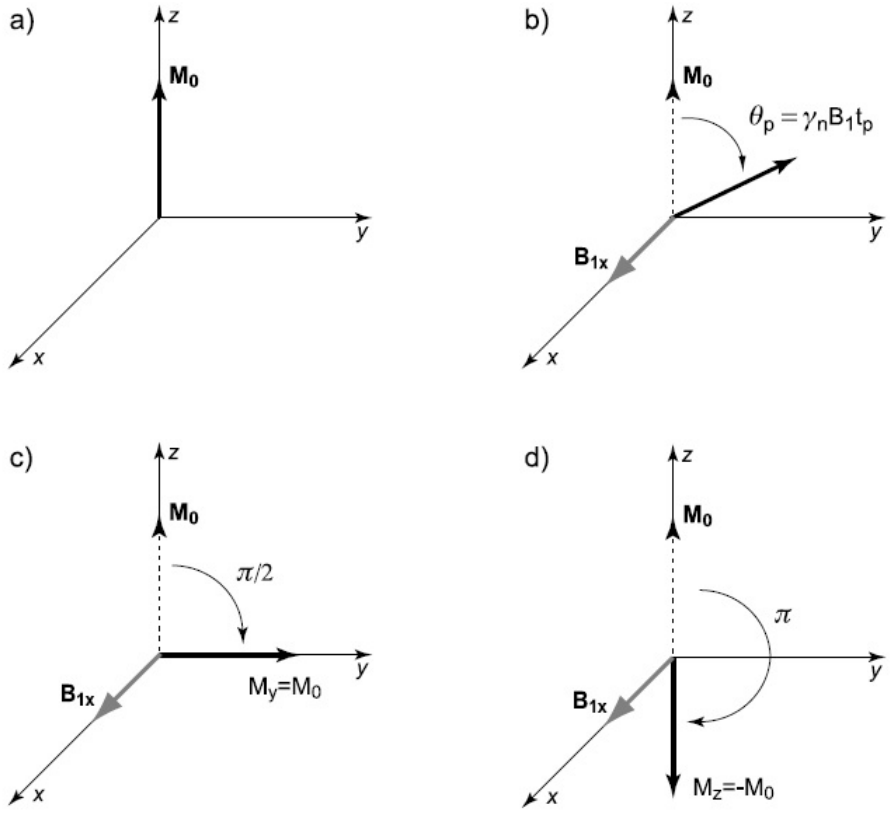


Figure 2.2: The effect of the RF field on nuclear magnetization.

pulse as shown in figure 2.2. This can be also thought as a spin-1/2 system. The $\pi/2$ and π will work as equalization and an inversion of population. The return to equilibrium needs the system give up some energy to the environment (generally named the lattice). This process is termed relaxation and is detailed in the next section.

2.1.3 Relaxation

After a $\pi/2$ pulse, the collection of nuclear spins precess around plane perpendicular to B_0. This is a non-equilibrium situation and after some time the magnetization returns to its equilibrium position. This thing is caused by two process which are occurring simultaneously namely transverse and longitudinal relaxation.

The transverse relaxation is the process which causes the disappearance of components of the nuclear magnetization M form axis perpendicular to B_0. Due to the spread

in frequencies of nuclear spins the transverse magnetization causes the dephasing in spins. This dephasing is known as *spin-spin relaxation*. In the spin-spin relaxation, the spins distribute randomly and making the net transverse magnetization zero. The differential equation corresponding to this is given by,

$$\frac{dM_{x,y}}{dt} = -\frac{M_{x,y}}{T_2} \tag{2.14}$$

where T_2 is the transverse or spin-spin relaxation time. The solution of this equation is given as,

$$M_{x,y} = M_0 e^{-t/T_2} \tag{2.15}$$

where M_0 is the initial value of the magnetization.

Whereas the longitudinal magnetization will tend to return to its equilibrium position. This restoring will need some energy to go to other level. There will be an exchanged energy between system and environment named the lattice. Due to this, it is known as *spin-lattice relaxation*. The differential equation is given as,

$$\frac{dM_z}{dt} = -\frac{M0 - M_z}{T_1} \tag{2.16}$$

where $T1$ is the spin-lattice relaxation time. The solution to such equation is given by,

$$M_z = M_0(1 - e^{t/T_1}) \tag{2.17}$$

The relaxation times are parameters characteristics of each particular system and magnitudes depend on factors such as temperature, physical state, magnitude of external magnetic field, etc.

2.1.4 Nuclear Spin Interactions

The nuclear spins are not at all isolated from each other and from the local environment. These interaction influence the exact value of the resonance frequency of each nucleus and its environment. During NMR experiment, the interactions between nucleus and the electromagnetic fields present in the environment. This can be well understood by the nuclear spin Hamiltonian ($\mathcal{H}_{nuclear}$) which can be written as,

$$\mathcal{H}_{nuclear} = \mathcal{H}_{ext} + \mathcal{H}_{int} \tag{2.18}$$

where $\mathcal{H}_{ext}$ represents the interactions of the nucleus with applied electromagnetic fields and $\mathcal{H}_{int}$ corresponds to internal interactions with the environment of nucleus. The external Hamiltonian can be written as,

$$\mathcal{H}_{ext} = \mathcal{H}_Z + \mathcal{H}_{RF} \tag{2.19}$$

There are several contributions to the internal Hamiltonian which depends upon the physical characteristics of the material. The internal Hamiltonian is given by,

$$\mathcal{H}_{int} = \mathcal{H}_{CS} + \mathcal{H}_D + \mathcal{H}_J + \mathcal{H}_Q \tag{2.20}$$

where $\mathcal{H}_{CS}$ is the chemical shift interaction of the nucleus with surrounding orbiting electrons; $\mathcal{H}_D$ is the dipolar interaction between nuclei; $\mathcal{H}_J$ is electron-mediated interaction and $\mathcal{H}_Q$ is the quadrupolar interaction for spin greater than 1/2. Now we will discuss the terms in internal Hamiltonian one by one.

Chemical Shift

The magnetic field experienced by the nuclei is not equal to the external magnetic field due to the presence of other orbiting electrons. The local magnetic field is given by,

$$B_{loc} = (1 - \widetilde{\sigma})B_0 \tag{2.21}$$

where $\widetilde{\sigma}$ is known as the chemical shift tensor. For an isotropic liquid substance, the average value of chemical shift known as isotropical chemical shift [IO07],

$$\mathcal{H}_{CS} \cong \gamma_n \hbar \sigma_{iso} B_0 I_z \tag{2.22}$$

where σ_{iso} is known as isotropical chemical shift which is related to trace of $\widetilde{\sigma}$.

There is a small correction added to the magnetic field due to the effect of the chemical shift. The shift in resonance frequency away from Larmor frequency is given by,

$$\omega = \omega_L(1 - \sigma_{iso}) \tag{2.23}$$

The resonance frequency is usually expressed as a relative shift measured with reference to the resonance frequency (ω_{ref}) of the standard substance,

$$\delta = \frac{\omega - \omega_{ref}}{\omega_{ref}} \tag{2.24}$$

where δ is called as the chemical shifts of the resonance lines which is usually expressed in ppm (parts per million).

J-Coupling

The J-coupling is also known as indirect or scalar coupling is an interaction between the nuclear magnetic dipole moments of neighbor nuclei. This interaction is mediated by electron cloud involved in the chemical bond. The Hamiltonian for this kind of interaction for two spins I_1 and I_2 for hetero-nuclear case is given by,

$$H_J = 2\pi\hbar J I_{1z} I_{2z} \tag{2.25}$$

When this Hamiltonian is taken into account as a perturbation in Zeeman Hamiltonian with chemical shifts, it is observed that each line split in a multiplet which depends on number of identical nuclei coupled. The J-coupling can be positive or negative means the coupling can favor either an anti-parallel or parallel alignment of nuclear spins.

2.1.5 NMR of Two Coupled Spins

Here we the NMR of two coupled spin-1/2 system. Consider two J-coupled spins denoted as 1 and 2. Taking the magnitude of J-coupling to be much smaller than the difference in resonant frequencies of the two nuclei this is for a hetero-nuclear system which is usually named as 'AX' system. The Hamiltonian in the hetero-nuclear system is given by,

$$\mathcal{H} = -\hbar\omega_1 I_{1z} - -\hbar\omega_2 I_{2z} + 2\pi\hbar I_{1z} I_{2z} \tag{2.26}$$

The resonant frequencies includes the effect of chemical shifts. I_{1z} and I_{2z} are the angular momentum operator in z-direction for both the spins. The energy levels associated with this Hamiltonian are given by,

$$E_{+1/2,+1/2} = \hbar\Big(- \frac{\omega_1}{2} - \frac{\omega_2}{2} + \frac{\pi J}{2}\Big) \tag{2.27}$$

$$E_{+1/2,-1/2} = \hbar\Big(-\frac{\omega_1}{2} + \frac{\omega_2}{2} - \frac{\pi J}{2}\Big) \tag{2.28}$$

$$E_{-1/2,+1/2} = \hbar\Big(\frac{\omega_1}{2} - \frac{\omega_2}{2} - \frac{\pi J}{2}\Big) \tag{2.29}$$

$$E_{-1/2,-1/2} = \hbar\Big(\frac{\omega_1}{2} + \frac{\omega_2}{2} + \frac{\pi J}{2}\Big) \tag{2.30}$$

These are the energy levels and transition between them are allowed according to the selection rules. So there are four peaks in the spectrum corresponding to the frequencies $\omega_1 \pm \pi J$ and $\omega_2 \pm \pi J$. And the separation between each doublet is $2\pi J$. The Hamiltonian and energy levels will vary according to the number of spins in the system.

2.1.6 Density Matrix Formalism

The most appropriate approach to describe the NMR phenomena involves the use of density matrix formalism from statistical mechanics. Here we will not have access to individual particles but on an ensemble. The density operator ρ is a collection of identical, independent nuclei. The expectation value of any observable over the ensemble is given by,

$$\langle A \rangle = Tr\{\rho A\} \tag{2.31}$$

In an NMR experiment, the observables of interest are the components of nuclear magnetization. These components are proportional to ensemble average values of nuclear spin operator. Taking magnetization in x-direction for example, the observable is given by,

$$\langle M_x \rangle \propto Tr\{\rho I_x\} \tag{2.32}$$

The time evolution density operator is given by the Liouville-von Neumann equation as,

$$\frac{d\rho}{dt} = \frac{\iota}{\hbar}[\rho, \mathcal{H}] \tag{2.33}$$

where $\mathcal{H}$ is the Hamiltonian of the system. If $\mathcal{H}$ is time independent, then ρ is given by,

$$\rho(t) = e^{-(\iota/\hbar)\mathcal{H}t}\rho(0)e^{(\iota/\hbar)\mathcal{H}t} \tag{2.34}$$

the term $e^{-(\iota/\hbar)\mathcal{H}t}$ is called the evolution operator. The above equation can be written as,

$$\rho(t) = U\rho(0)U^{\dagger} \tag{2.35}$$

When the Hamiltonian is not time-independent then it can be split into a finite number of time-independent terms. And it is useful in understanding the NMR experiments which consists of sequence of RF pulses and free evolution time periods.

In an orthonormal basis, the density operator can be represented by a matrix, known as density matrix,

$$\rho = \begin{bmatrix} \rho_{11} & \rho_{12} & \rho_{13} & \cdots & \rho_{1n} \\ \rho_{21} & \rho_{22} & \rho_{23} & \cdots & \rho_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho_{n1} & \rho_{n2} & \rho_{dn3} & \cdots & \rho_{nn} \end{bmatrix}$$

This density matrix must satisfy some of the requirements such as,

- It should be a Hermitian operator.
- The diagonal elements must be positive or zero.
- The trace should be unity.

The diagonal elements in the density matrix represents the *population* and off diagonal elements are *coherence*. The element ρ_{mm} gives the probability of finding a member of ensemble in that state of quantum number m. The coherence term is basically due to the relaxation phenomena i.e. transverse magnetization and population are related to longitudinal magnetization.

2.2 NMR Quantum Information Processor

2.2.1 Qubits in NMR

The basic requirement for qubit in any system is that they must be well characterized and susceptible to manipulation by external perturbation. So that one can be able to control them. A natural implementation in NMR of a qubit is an isolated spin 1/2 in a magnetic field. Here we represent the general state as,

$$|\phi\rangle = \alpha\,|+1/2\rangle + \beta\,|-1/2\rangle \tag{2.36}$$

The state $|+1/2\rangle$ labeled as $|0\rangle$ and state $|-1/2\rangle$ as $|1\rangle$.

For spin 1/2 system each qubit is associated to a spin. A system of n qubits can be implemented by as system on n coupled spins 1/2, with the following Hamiltonian,

$$\mathcal{H} = -\sum_{i}^{n} \hbar\omega_i I_z^i + \hbar 2\pi \sum_{i\neq j}^{n} J_{ij} I_{iz} I_{jz} \tag{2.37}$$

It is difficult to obtain many qubits in NMR liquid state as they require a sample with n NMR distinguishable spins in a single molecule, which is very difficult to obtain.

In NMR, system is not constituted of single molecule but an ensemble of molecules. So one can think of NMR liquid sample as constituted by a huge number ($\approx 10^{23}$) of molecular quantum processor executing a kind of parallel processing.

2.2.2 DiVincenzo Criteria

The DiVincenzo criteria [DiV00] is a list of conditions that are necessary for constructing a quantum computer on a real system. There are total seven criteria, first five are for quantum computation and rest two are for quantum communication. We will discuss on first five criteria. The NMR system very successfully follows these criteria. The criteria are listed below-

- **Characterizing Quantum system-** In an NMR system the states are well characterized by density matrices.
- **Initialization-** NMR is different from other systems where initialization is done using pure states, here initialization is done using pseudo pure states. There are several methods of preparing the pseudo pure state and will be discussed in next section.
- **Implementation of Quantum Gates-** The quantum gates can be implemented by the RF pulses in an NMR system
- **Measurement-** in NMR we are able to measure states and the measurement results are processed by quantum state tomography.
- **Decoherence-** This is what limits the computation. In NMR, decoherence time is order of milliseconds to seconds depending upon the sample and one can do the computation in this range.

In the next section we will understand the some of the basic things which are needed for the physical realization of an NMR quantum information processor.

2.2.3 Preparation of Pseudo Pure States

The fact that NMR experiment are only sensitive to the traceless deviation density matrix makes the idea of pseudo-pure states [DGCH97]. So, one might search for transformations that are applied to thermal equilibrium density matrix to produce a deviation density matrix with the same form as a pure state.

The density matrix corresponding to a pure state follows the following properties: $\rho = \rho^n$ and $Tr(\rho^2) = 1$. For a mixed state it follows: $\rho \neq \rho^n$ and $Tr(\rho^2) < 1$. After applying some unitary transformation the state transforms as $\rho^{'} = U\rho U^{\dagger}$. And after checking by trace condition it follows that it is not possible to obtain a pure state from a mixed state by only using unitary transformations.

As discussed to prepare a pseudo pure state we need something more than a unitary transformation such as non-unitary rotations etc. The methods to produce a pseudo pure state is described here.

Temporal Averaging

In this method of preparation of pseudo pure states, states are prepared by applying unitary transformations to a thermal state and are combined to produce an average state that behaves like a pure state in NMR.
Let us consider a two-qubit system density matrix in the computational basis $|00\rangle$,$|01\rangle$, $|10\rangle$,$|11\rangle$ as,

$$\rho_i = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & d \end{bmatrix}$$

Lets apply the unitary transformation U_0, U_1 and U_2 on ρ_i which are given by,

$$U_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$U_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & d \end{bmatrix}$$

$$U_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \end{bmatrix}$$

where U_0 is an identity operator, U_1 can be get by applying the two CNOT gates and U_2 is conjugate transpose of U_1. The final resulting density matrices ρ_0, ρ_1 and ρ_2 will be obtained after the action of unitaries. Taking the average over these states to obtain the effective state,

$$\rho_{00} = \rho_0 + \rho_1 + \rho_2 = \begin{bmatrix} 3a & 0 & 0 & 0 \\ 0 & b+c+d & 0 & 0 \\ 0 & 1 & b+c+d & 0 \\ 0 & 0 & 0 & b+c+d \end{bmatrix}$$

since $b + c + d = 1$, so the matrix will change and we will get the final density matrix as,

$$\rho_{00} = (1-a)\mathbb{1} + (4a-1)\left|00\right\rangle\left\langle 00\right| \tag{2.38}$$

The first term on right side is proportional to the identity and is not detected in NMR and neither affected by the RF field. The second term transform as under the action of RF pulse and also contribute to detecting signal. Hence, this state will act like a pure state. The other pseudo pure states can also be obtained by applying some unitary operations.

This can be generalized to system with large number (n) of spins. In that case it is necessary to combine $2^n - 1$ prepared states to create a pseudo pure state.

Spatial Averaging

The spatial averaging technique is based on dividing the system in spatially separated sub-ensembles. These sub-ensembles can be accessed independently in NMR using combination of RF pulses and pulsed gradients. The pseudo pure state obtained

will be averaged over all the sub-ensembles. This is a single shot implementation for pseudo pure state. The unitary for getting pseudo pure state ρ_{00} is given as,

$$U = \Big[G_z(\tau)\Big(\frac{\pi}{4}\Big)^{I_1}_{-y} U_J\Big(\frac{1}{2J}\Big)\Big(\frac{\pi}{4}\Big)^{I_1}_{x} G_z(\tau)\Big(\frac{\pi}{3}\Big)^{I_2}_{x}\Big] \tag{2.39}$$

where $G_z(\tau)$ represents the gradient pulse for duration τ.

2.2.4 Generating Quantum Gates using RF Pulses

Quantum computation is based on a set of universal logic gates and they are nothing but the unitary operations. In NMR, manipulation of spin states through unitary transformation using RF pulses or evolution under internal interaction is possible.

The most elementary single-qubit operation which performs a rotation on single spin and for which the rotation operator is given by,

$$R_{\hat{n}} = exp(-\iota\theta n.I) \tag{2.40}$$

where n is the unitary vector which defines the rotation axis, θ is the rotation angle and I is the nuclear spin operator which is given by,

$$I = I_x\hat{i} + I_y\hat{j} + I_z\hat{k} \tag{2.41}$$

Since RF pulses are direct implementation of unitary operations, so the action of on resonance pulse with phase ϕ and pulse duration t_p is described by a pulse propagator which is given by,

$$(\theta)^I_\phi = exp(-\iota\Omega t_p I_\phi) = exp(-\iota\theta I_\phi) \tag{2.42}$$

where,

$$I_\phi = I_x cos(\phi) + I_y sin(\phi) \tag{2.43}$$

and $\theta = \Omega t_p$. This shows that any spin rotation in xy plane can be generated by RF pulses of proper phase, amplitude and duration.

Some of the single-qubit gates which can easily be generated using RF pulses are,

- **NOT Gate-** It can be generated via the π-pulse along the x-axis.

$$(\pi)_x^I = exp(-\iota\pi I_x) = e^{-\iota\pi/2}\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} \tag{2.44}$$

- **Phase Gate-** This gate can be achieved from a z-rotation of arbitrary flip angle.

$$R(\theta) = \left(\frac{\pi}{2}\right)_x^I (\theta)_y^I \left(\frac{\pi}{2}\right)_{-x}^I = e^{-\iota\theta/2}\begin{bmatrix}1 & 0\\ 0 & e^{\iota\theta}\end{bmatrix} \tag{2.45}$$

  when $\theta = \pi/4, \pi/2$ and π we get T, S and Z gates respectively.

However in the case of multiple qubits the situations somewhat changes as now the spins will interact with each other. For a hetero-nuclear case where different spins have distinct NMR frequencies and are J-coupled with each other. In this case we can apply a resonant pulse to one of them without affecting the other and if pulse amplitude is much higher than the magnitude of J-coupling then we can neglect the effect of that. These RF pulses act as a selective pulse to each spin and these are known as *hard pulses* as the RF power used is high.

In the case of homo-nuclear system, where each nuclei have a close resonance frequency to the other nuclei and to perform rotation of single spin it is necessary to use RF pulses of narrow excitation profiles. These pulses have long durations and low power so these are known as *soft pulses*. But the error associated with them is because of their long duration so we can not neglect the evolution under J-coupling which is not a desirable feature. The solution to this is apply a self-refocusing pulse or setting up the pulse duration to be multiple of J-coupling evolution period. But all these factors will only increase the time for pulse sequence which can lead to other errors.

The construction of multiple qubit gates can be done using RF pulses and delays. There are many different methods to generate the pulse sequence for these gates as we will discuss in next chapter. Here we designed a new method for generating the pulse sequence by optimization using genetic algorithms. In the upcoming chapters this new method is introduced with results for some of the very important gates in quantum computation and quantum information.

Chapter 3

Optimization of Quantum Gates

"Computer programs that evolve in ways that resemble natural selection can solve complex problems even their creators do not fully understand"

— J.H.Holland

3.1 Introduction

In NMR quantum information processor, to implement any process we apply unitary operations and to apply unitary operations one need to break these operations into pulse sequences. The pulse sequence in NMR system is applied through RF fields. The process of breaking a unitary into a pulse sequence is not an elementary task. So we consider this as an optimization problem to reach an optimum solutions of pulse sequence to get the desired unitary process done.

Since an NMR quantum information processor is prone to different types of error just like any other physical realization of quantum information processor. So some times the processes are not as efficient as we want them to be. This loss of fidelity is due to the errors associated with offset frequency, inhomogeneity, refocusing, decoherence etc. The main task here was to design an optimization algorithm to break down the multiple qubits unitary into the high fidelity pulse sequences.

There were several attempts in development of pulse sequence such as, break down of unitary into two qubit CNOTs and one qubit rotation gates [JAS96], GRAPE algorithms [NK05], using transition pulses [XF02], using genetic algorithms [MK12][AA09], Bang-Bang [GBM16] and many more. Some of the attempts are very successful in

physical realization also. Here we developed an optimization scheme for pulse sequence using genetic algorithms. This algorithm break down the unitary into pulse sequence by using only hard pulses and delays. The experimental results were also obtained with high fidelity.

In the next section, we will try to understand the biologically inspired genetic algorithm and how it can be used in optimization schemes.

3.2 Genetic Algorithm

3.2.1 Introduction

Genetic algorithm, which is a global optimization technique which mimics the behaviour of biological evolution in nature. It was developed by John Holland in 1970 [Hol92]. It is the subset of larger class of evolutionary algorithms. Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection [Mit96].

The genetic algorithms are different from other classical algorithms which uses local search optimization techniques in the following ways-

- Classical algorithm generates a point at each iteration and this point will lead to a solution. Whereas, genetic algorithm generates a population at each iteration and the best out of them will reach to a solution.
- In classical algorithms the next point is selected deterministically but in genetic algorithms the next population is selected randomly.

It is able to solve both constrained and unconstrained optimization problems very efficiently. It has been used in variety of fields and the range of applications of the genetic algorithm are nearly boundless. From antenna design [G.S06] to the development of solar tracking systems to maximize power harvesting [S.M08]. From the development of muscle based control methods for bipedal creatures [T.G13] to the optimization of molecular geometry [D.M95].

The genetic algorithms exploits the population (chromosomes) of candidate solutions to an optimization problem to evolve towards a better solution. Each candidate

has a set of properties (genotype) which can be altered or mutated. There are some of the important features of algorithm which needs to be understood before designing it for the purpose of pulse sequence development. The next sections will focus on these features.

3.2.2 Initialization

Once the genetic representation is defined we need to initialize the problem by giving population size. Initialization is a crucial step for the algorithm since the optimality of the solution depends on this. Typically the initial population vary from a range of hundreds to thousands but it heavily depends on the search space of the problem. Often initial population is generated randomly so that it can cover the whole search space and sometimes when we know where the solution exist the initial population seeded according to that.

3.2.3 Fitness Function

Fitness function is a special type of objective function that is used to evaluate that how close a given solution form a desired solution. It basically checks the overlap between a given solution and desired solution. The fitness function depends upon the type of problem one is addressing.

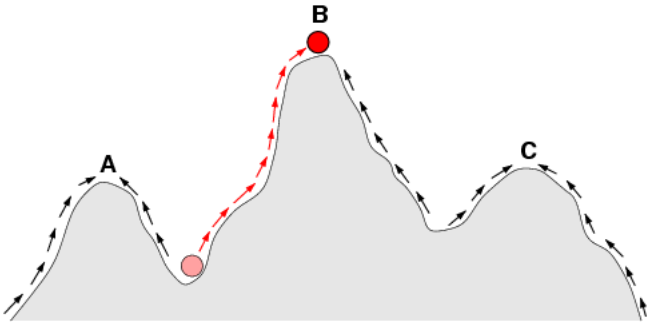


Figure 3.1: Figure of fitness landscapes. Arrows indicates the flow of population. Points A and C are local optima and B is global optima.

The main aim of the genetic algorithm is to find the global optimum in a given workspace for a given function. This optimum may either be a maximum or a minimum. This choice rests solely on the user and the nature of the fitness function. A way of looking the fitness function is in terms of fitness landscape as shown in figure 3.1. We have observed that,

- The steeper the fitness landscape towards the global optimum, the harder it will be for the algorithm to find it.
- The presence of local optima seek to divert the genetic algorithm to a state of stagnation. Hence the phenomenon of fewer local optima in the fitness landscape is desired.

3.2.4 Selection

During each successive iteration or generation some of the individuals are selected based on their fitness to breed a new generation. It is based on the criteria of *'survival of the fittest'*. Since the higher fit individual is more likely to get selected to breed. The two main features which determine the direction the algorithm takes in the fitness landscape are,

- **Selection pressure-** It influences the likelihood of the algorithm to move towards or away from the local optima. Higher selection pressure [Bac94] chooses more fit individuals for the processes breeding, thereby allowing extremely fit individuals to get even fitter. This situation is useful when the algorithm has almost reached the global optimum. On the other hand, selection pressure of lesser magnitude allows for the algorithm to branch out and reach all areas of the workspace. If the range of fitness values among the population is high, the lower selection pressure will ensure that the algorithm does not get stuck on local optima.
- **Stochastic Noise-** As the term denotes, stochastic noise adds a random amount of noise to ensure that the algorithm does not stagnate at any of the local optima. If there are a large number of pockets of local optima, higher stochastic noise will move larger amounts of the population away from the local optima. The downside here is that even if a solution representing the global optimum is found, the stochastic noise could make the algorithm take a wrong decision and hence move away from the global optimum.

There are several selection methods [KJ13] such as Roulette wheel selection, stochastic universal sampling and linear ranking etc. One can use the existing selection methods or can generate their own method depending on the nature of the problem.

3.2.5 Genetic Operators

The next thing after selection is the mixing of the population to create new generation. The mixing of population is done so that the problem aim towards a solution. The solution can be reached in such a way that the next generation will have different chromosomes than the previous generation and will have a average high fidelity in this way after some generations the algorithm will reach to an optimal solution. There are several operators for this but some of them are inspired by genetics itself. Here are the two operators which are widely used in genetic algorithm,

Crossover

The crossover is used to vary the genotypes of form one generation to the next. It is similar to the crossover in biological systems where two individuals are selected to act as a role of parents and the chromosomes of them recombined to produce an offspring.

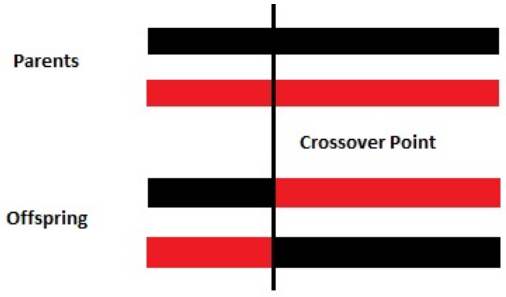


Figure 3.2: Crossover operator in genetic algorithm

The steps involved in the crossover operation figure 3.2 are,

- Two members are chosen from the population based on one of the selection methods introduced above.
- A locus (crossover point) point is chosen at the same point in both the individuals.
- The portions after the locus are swapped.

Mutation

This is a genetic operator which is used to maintain the genetic diversity in the population so that the population does not stagnate at a point. Genetic algorithm can

come to a better solution using mutation. Mutation involves randomly picking of an individual and changing its genotype randomly as shown in figure 3.3.

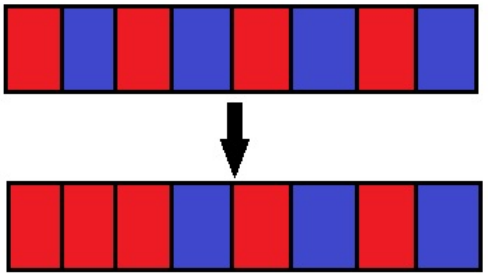


Figure 3.3: Mutation operator in genetic algorithm

There is always a probability associated with the mutation. The mutations could either be good or bad. But after the mutations, only the individuals with a higher probability of survival, reproduced and their mutations persevered. Similar to that, we introduce a stochastic noise in our system which implements the function of mutation. An individual is chosen at random , and according to its representation scheme, a few of its variables are changed.

3.2.6 Termination

The algorithm runs until some termination condition is being given. The termination condition depends upon the representation schemes and the type of the problem one is dealing with. The most common used termination condition is when we the desired solution is obtained or number of generations passed the limit of maximum generations.

The next section will give an idea of how we incorporated the genetic algorithms in solving the problem of pulse sequence development with great efficiency.

3.3 Optimization using Genetic Algorithms

3.3.1 Introduction

As we have seen in the previous section that how genetic algorithm works and how it reaches an optimal solution. This global optimization scheme is very efficient and time saving. Here we seek to optimize quantum gates for implementing it on an NMR quantum information processor. We need to first design an algorithm according to the problem, the next step will be to incorporate genetic algorithm to solve this problem.

The liquid state NMR system Hamiltonian for a n spins system is composed of terms which describes the interaction of spin with external magnetic field and interactions of spins with each other. The rotating frame Hamiltonian is given by,

$$\mathcal{H}_{\text{NMR}} = -\pi \sum_{i=1}^{n} (\nu_i - \nu_{rf}^i)\sigma_z^i + \sum_{i<j,=1}^{n} \frac{\pi}{2} J_{ij}\sigma_z^i\sigma_z^j. \tag{3.1}$$

where ν_i are the chemical shifts, ν_{rf}^i are the rotating frame frequencies, J_{ij} are the scalar couplings between the spins and σ_z are the Pauli-z matrix which is related to angular momentum operator (I) as,

$$I_z = \frac{\sigma_z}{2} \tag{3.2}$$

For designing pulse sequences we exploited the NMR system Hamiltonian itself.

To proceed further and to make use of genetic algorithm in this problem we need to define a fitness function. The fitness function we will used here is fidelity function which is most commonly used in quantum computation and quantum information. The fidelity function (F) [NK05] is given by,

$$F = \left| Tr[(U_{opt})(U_{tar}^{\dagger})] \right|^2 \tag{3.3}$$

where U_{tar} is the target unitary operator which is desired and U_{opt} is the derived unitary operator. It is normalized such that when U_{opt}=U_{tar}, the fitness is 1.

The derived unitary operator is the one we are optimizing and taking it close to target unitary. This fidelity function will give some value for every optimized unitary

which tells us how much it is overlapped with the target unitary. The derived unitary operator (U_{opt}) is defined as,

$$U_{opt} = \prod_{l=1}^{N} exp[-i(\mathcal{H}_{\mathrm{NMR}} + \Omega I_{\phi_l k})\tau_l] exp[-i\mathcal{H}_{\mathrm{NMR}}\delta_l] \tag{3.4}$$

where,

$$I_{\phi_l} = \frac{1}{2}(\sigma_x \cos\phi_l + \sigma_y \sin\phi_l) \tag{3.5}$$

σ_x and σ_y are the Pauli-x and Pauli-y matrices respectively. The equation 3.4 describes the time evolution of Hamiltonian. The first term of the equation shows the time evolution of system and RF field Hamiltonian and second term is related to time evolution of system Hamiltonian in absence of RF field. The Ω is the power of the pulse, τ is the width of the pulse and δ is delay between the pulses as shown in the figure 3.4.

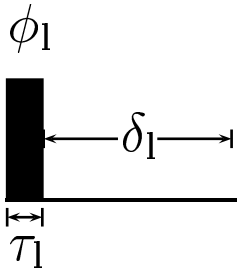


Figure 3.4: Propagator represents the l^{th} pulse of τ_l width along phase ϕ_l followed by delay δ_l.

The main idea is to develop a pulse sequence comprising of hard pulses and delays. The term hard pulse is used because the RF field is greater than the size of reduced field and the time for implementing them is less. This is done in order to minimize the errors which results in a high experimental fidelity. To achieve this we fixed the value of Ω to its full power and varied pulse width (τ), phase (ϕ) and delay (δ). So this will act like a propagator as shown in figure 3.4 which will repeat itself N times to reach a solution. The three parameters which are explained above were exploited to reach an optimal solution for the pulse sequence.

3.3.2 Representation Scheme

Every problem which needs to be solved with the genetic algorithm should be represented in an efficient way. The optimality of the solution depends upon the representation scheme as well. In our optimization problem, each pulse sequence was represented as a matrix of order $N \times 4$ where N represents the number of operation needs to be done in order to reach a desired unitary. Every row here represents a hard pulse and a delay just like a propagator as shown in figure 3.4. The detailed description of every column is given below,

- **Column 1-** This column represents the pulse width τ of a hard pulse. Since we are keeping the power (Ω) full so the angle (θ) of the hard pulse which is $\theta = \Omega\tau$ will vary by varying τ. Since θ is restricted in a range of $\{0, 2\pi\}$ so width (τ) will be adjusted accordingly.

- **Column 2 and 3-** These columns are basically associated with phase (ϕ) of the pulse. Since ϕ is restricted in a range of $\{0, 2\pi\}$ when taken in radians, we divided it into two parts i.e. divided the search space to make it easy. The range is divided into two parts as $\{0, \pi\}$ and $\{-\pi, 0\}$. The second column will have entries 0 or 1 which represents from which set do they belong and third column will have values in range $\{0, 180\}$.

- **Column 4-** This column contains the value of delays (δ) between the hard pulses. The values actually depends upon which unitary are we optimizing. The values of delays are typically in range of microsecond to second.

In every pulse sequence the number of rows (N) will vary depending upon which unitary is being optimized. Sometimes the number of hard pulses and delays are less and sometimes more. The increase in number of rows increase the control over the system. So giving a variable number of rows would be a good idea to start the algorithm.

3.3.3 Selection

The selection of individuals for the processes of crossing over and mutation is of major importance, as it describes the direction taken by the population in the fitness landscape [Bac94]. Hence selection pressure at every generation is also of great importance. We thus arbitrated that in the initial stages, a low selection pressure would

be used to allow exploration of the candidate solutions in the workspace. If a potentially viable solution was recognized, the intensity of the selection pressure would be increased in order to allow for exploitation of neighbours of the recognized solution.

After attempting existing selection mechanisms as described previously we devised our own selection mechanism called Luck-choose which allowed the algorithm to converge to a solution much faster. It's mechanism of operation involved first multiplying pseudo-randomly generated weights to the fitness values of all individuals, and subsequently determining the highest among the output values.

Now as the representation scheme and selection method is set we need to define some genetic operators which will help in increasing the genetic diversity which will directly help to get an optimal solution.

3.3.4 Crossover

As previously seen that how crossover which is inspired by genetics works. These operations should be modified according to the representation scheme. In a matrix representation scheme the crossover operation works as follows,

- Two members are chosen from the population using Luck-choose selection process.
- Two numbers are randomly chosen within the maximum number of rows, and two numbers are randomly chosen within the maximum number of columns.
- The first number of each corresponds to the starting point of crossover and the second number corresponds to the ending point.
- Using the above four numbers we will be able to create a rectangular sub-matrix inside a matrix. This sub-matrix of elements is swapped between both the selected individuals.

One example of this kind of operation is as shown in the figure 3.5.

3.3.5 Flip

This is another kind of operation which was used in the optimization scheme. It exists due to existence of the non-commutativity of each propagator with the other. It works in the following way,

- One member is chosen from the population using Luck-choose selection process.
- Two random rows are chosen from the given number of rows.
- These chosen random rows swapped with each other.

The example for this kind of operation is as shown in the figure 3.5.

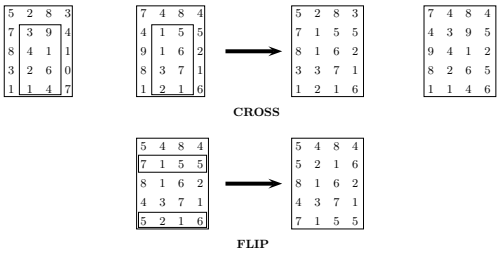


Figure 3.5: Crossover and flip operations.

3.3.6 Mutation

This operation depends heavily on the amount of stochastic noise required in the system as explained earlier. Stochastic noise adds a random amount of noise to ensure that the algorithm does not stagnate at any of the local optima. In the initial stages, low stochastic noise is preferred, so the mutation operation may be disabled. But after the population explores the workspace through a few generations, the chances of getting stuck in local optima increase, and hence the probability of occurrence of the mutation operation is increased in steps, until a threshold value, above which the stochastic noise would only serve to drive candidate solutions away from the global optimum.

Mutation works in the following way- it takes a single member of population via luck chose selection method, and changes all the data values of the chosen member.

The algorithm was designed as explained above. The algorithm terminates either when it reaches an optimal value or when the number of generations crosses the maximum number of generations.

3.4 Optimization Details

The algorithm was used to optimized three qubit unitary matrices. The implementation of the algorithm with the above modifications was done using MATLAB [MAT15] and the flow chart is as shown in the figure 3.6. After running the genetic algorithm, outputs are obtained with fidelity in the lower 80's. Hence they are passed through a local optimizer to obtain more precise outputs. This local optimizer randomizes the values of the first, third and fourth columns within a small range of values around the existing value. Crossover operations are performed again, to further increase efficiency of optimization and outputs can be derived with fidelity above 0.99 by this method.

An iteration of the program running the algorithm for 15 rows and 500 chromosomes, took an average time of 3 hours using a single core for processing, on an i7-4700MQ processor with 8 GB of RAM. For parallel processing, the parallel computing toolbox was used, enabling us to run 6 iterations simultaneously on 6 virtual cores for approximately 4 hours. This reduced the average run-time per iteration to approximately 40 minutes. The local optimizer however could be run from 10 minutes to 15 hours depending on the final fidelity required and the fidelity of the starting matrix.

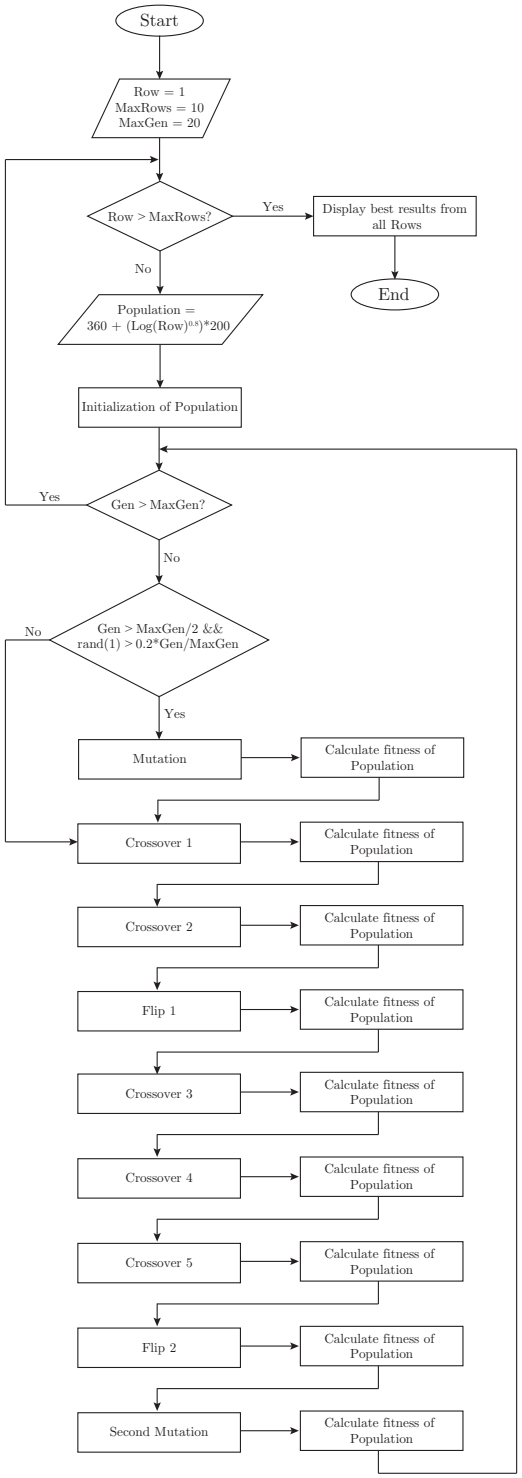


Figure 3.6: Flow chart for optimization scheme

Chapter 4

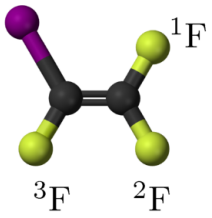
Experimental Implementation of Optimized Quantum Gates

4.1 Introduction

The three qubit unitary matrices were optimized using the optimisation technique explained previously. Since the technique we have shown here can be used for any number of qubits but we have shown the results for three qubit gates. From the optimization, we got very high fidelity pulse sequence for unitary matrices. The optimized pulse sequences was experimentally implemented on an NMR quantum information processor with very high experimental high fidelity.

4.2 Experimental NMR Qubits

The three fluorine (^{19}F) of molecule iodotrifluoroethylene are used to encode the three qubits. The molecular structure with chemical shifts (ν_i) and scalar couplings (J) are as shown in the figure 4.1.



ν_{1_F}=34011.92Hz J_{12}=69.65Hz
ν_{2_F}=43479.26Hz J_{13}=47.67Hz
ν_{3_F}=57296.29Hz J_{23}=$-$128.32Hz

Figure 4.1: Molecular structure of iodotrifluoroethylene with the measured chemical shifts (ν_i) and scalar couplings (J)

All experiments were performed at 294 K on a 400 MHz NMR spectrometer. The fluorine spin resonates at a Larmor frequency of 376.45 MHz. The rotating frame Hamiltonian of the three qubit system can be written as,

$$\mathcal{H}_{\text{NMR}} = -\pi \sum_{i=1}^{3} (\nu_i - \nu_{rf}^i)\sigma_z^i + \sum_{i<j,=1}^{3} \frac{\pi}{2} J_{ij}\sigma_z^i\sigma_z^j. \tag{4.1}$$

where ν_i are the chemical shifts, ν_{rf}^i are the rotating frame frequencies of i^{th} spin, J_{ij} are the scalar coupling constants between i^{th} and j^{th} spin and σ_z are the Pauli-z matrix which are related to angular momentum operator (I) as $\sigma_z = I_z/2$.

For the experiment, the system we are chose was a homo-nuclear system since the same three nuclei are being used as three qubits. So the all three spins will be controlled simultaneously. In this case the optimized unitary (U_{opt}) will change as follows,

$$\begin{aligned} U_{opt} = &\prod_{l=1}^{N} exp[-i(\mathcal{H}_{\text{NMR}} + \Omega(I_{\phi_l 1} + I_{\phi_l 2} + I_{\phi_l 3}))\tau_l] \\ &exp[-i\mathcal{H}_{\text{NMR}}\delta_l] \end{aligned} \tag{4.2}$$

In NMR experiment, the liberty to apply any pulse through any ϕ axis was exploited. So, the phase ϕ was fixed in a range of $\{0, 2\pi\}$ in radians. The power (Ω) of hard pulse was kept fixed at 120.88×10^3 rad/s, the hard pulse angle was taken in range of $\{0, 3\pi/2\}$ and range for length of the pulse (τ) was adjusted according to these two factors. The delay (δ) factor between the pulses was also taken in some range depending upon which unitary is being optimized. So in the beginning we have to start with a guess or some idea for the range of time factors.

For initialization, the three-qubit system was prepared into a pseudo-pure state $|110\rangle$ via the spatial averaging technique whose density operator is given by,

$$\rho_{110} = \frac{1-\epsilon}{8} I + \epsilon \left|110\right\rangle \left\langle 110\right| \tag{4.3}$$

where thermal polarization (ϵ) is approximately 10^{-5} and I is a 8×8 identity matrix. The experimentally created pseudo-pure state was tomographed with a fidelity of 0.97. All the experimental density matrices were reconstructed using a reduced

tomographic protocol and maximum likelihood estimation [HS16], with the set of operations given by $\{III, IIY, IYY, YII, XYX, XXY, XXX\}$; which are sufficient to determine 63 variable for three-qubit system. Where I is the identity operation, X and Y are the single spin operator which can be implemented by applying a $\pi/2$ pulse on the corresponding spin. All the operators for tomographic protocols were numerically optimized using genetic programming, each having a length of approximately 200 μs and had an average fidelity of ≥ 0.99. The fidelity of experimental density matrix was computed by measuring the projection between theoretical expected and experimentally measured states using the Uhlmann-Jozsa fidelity measure given by,

$$F = Tr(\sqrt{\sqrt{\rho_{theory}}\rho_{expt}\sqrt{\rho_{theory}}}) \tag{4.4}$$

where ρ_{theory} and ρ_{expt} denote the theoretical and experimental density matrices respectively.

4.3 Results

This section contains the optimized pulse sequence and experimental results for three qubit unitary matrices. The results for these optimized unitary gates are given in a tabular form for different gates, every row of table forms a propagator and number of rows represents the number of propagators to be joined together to form a pulse sequence.

4.3.1 90^o selective pulse

In a homo-nuclear system, the applied operations will act simultaneously on all the three spins. So, to rotate a single spin we need a selective excitation pulse. This kind of pulse is used in quantum state tomography and other phenomenas. Here we optimized the pulse sequence via genetic algorithms for 90^o selective pulse on third qubit along Y-axis. The pulse sequence contains only hard pulses and delays. The unitary for the selective pulse is given by the matrix,

$$U_{tar} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The optimized pulse sequence for this unitary matrix is given in Table 4.1. The pulse sequence was obtained with a theoretical fidelity of 0.998 and the pulse duration corresponding to this unitary is $200\mu s$.

l	$\tau_l(\mu s)$	ϕ_l	$\delta_l(\mu s)$
1.	12	97.32	27
2.	36	17.26	8
3.	5	172.1	14
4.	36	328.2	27
5.	33	12.6	2

Table 4.1: Table representing the pulse sequence for selective pulse. First column represents the number of propagators to be joined together to form a pulse sequence. The second, third and fourth column represents the pulse width (τ), phase (ϕ) and delay (δ) respectively.

This pulse sequence was experimentally implemented on an NMR quantum information processor. The spectra for this is as shown in the figure 4.2. The figure shows what was desirable i.e. the excitation of only the third spin.

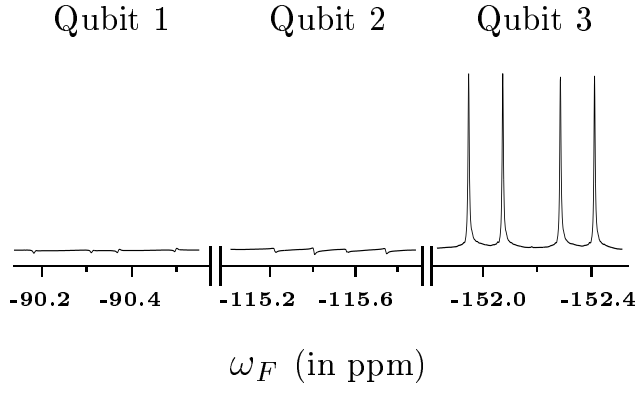


Figure 4.2: Experimentally implemented 90^o selective pulse on third spin along Y-axis.

4.3.2 Toffoli Gate

Toffoli gate which is commonly known as CCNOT gate is one of the important gates in quantum computation, as it is a universal gate for computation i.e. it can simulate any classical irreversible gate. This ensures that quantum computers are capable of performing any computation which is a classical computer may do.

Here we optimized this three-qubit gate using genetic algorithms which only incorporated hard pulses and delays. The first two qubits of this gate was considered as control qubits whereas the third qubit as a target. The unitary matrix corresponding to this gate is given by-

$$\mathrm{U}_{tar}= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The optimized sequence for this gate is as shown in Table 4.2. The optimized pulse sequence was obtained with a fidelity of 0.995 and with a pulse duration of 27ms. The pulse sequence was experimentally implemented on an initially prepared pseudo-pure state $|110\rangle$. The final state was obtained as $|111\rangle$ as theoretically predicted with an experimental fidelity of 0.93. The experimental tomographed results are as shown in figure 4.3.

4.3.3 Implementation of Fredkin gate

Fredkin gate which is commonly known as CSWAP gate is a universal gate for classical reversible computation. This gate is of interest because of its direct applications in error correction, cryptography, measurements, polarization transfer in NMR and some of the quantum algorithms. It was previously implemented by breaking it down into five two qubit gates. There are attempts to implement it in three transition pulses in NMR quantum information processor. But these attempts led to errors which led to decrement in experimental fidelity. Here we optimized this gate in single shot i.e. without breaking it down into other unitary using genetic algorithms which

l	$\tau_l(\mu s)$	ϕ_l	$\delta_l(\mu s)$
1.	32	243.22	539
2.	27	138.58	546
3.	39	2.47	499
4.	36	320.89	3488
5.	32	352.29	2495
6.	34	355.84	536
7.	37	175.98	1938
8.	29	20.45	1957
9.	34	354.75	542
10.	18	297.71	564
11.	15	215.55	2487
12.	27	308.2	550
13.	32	326.82	513
14.	13	122.09	541
15.	4	332.61	2518
16.	24	354.12	546
17.	36	310.6	3806
18.	32	210.97	1971
19.	30	3.74	504
20.	38	338.48	565

Table 4.2: Table representing the pulse sequence for Toffoli gate. First column represents the number of propagators to be joined together to form a pulse sequence. The second, third and fourth column represents the pulse width (τ), phase (ϕ) and delay (δ) respectively.

incorporated only hard pulses and delays. The Fredkin gate works in a way that the first qubit acts as a control qubit and if control qubit is 1, then the other two qubits swap their values. The unitary matrix corresponding to this gate is given by,

$$\mathrm{U}_{tar}= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The optimized pulse sequence for this gate is as shown in Table 4.3. The optimized pulse sequence was obtained with fidelity 0.99 and a pulse duration of 32ms.

l	$\tau_l(\mu\mathbf{s})$	ϕ_l	$\delta_l(\mathbf{ms})$
1.	29	108.63	1.584
2.	20	200.75	3.712
3.	11	258.7	0.639
4.	25	197.31	1.075
5.	9	172.69	4.568
6.	19	188.99	1.964
7.	31	265.02	4.416
8.	27	100.22	2.161
9.	12	86.57	3.276
10.	32	233.09	2.194
11.	10	186	1.569
12.	30	170.75	1.592
13.	17	4.55	3.269
14.	21	188.17	2.184
15.	28	36.37	2.152
16.	21	330.83	4.423
17.	7	46.59	2.194
18.	14	102.17	3.066
19.	28	295.24	1.574
20.	13	126.96	3.720

Table 4.3: Table representing the pulse sequence for Fredkin gate. First column represents the number of propagators to be joined together to form a pulse sequence. The second, third and fourth column represents the pulse width (τ), phase (ϕ) and delay (δ) respectively.

The pulse sequence was experimentally implemented on initially prepared pseudo-pure state $|110\rangle$. The output state was obtained as $|101\rangle$ with a fidelity of 0.96. The experimental tomographed results are as shown in figure 4.3b.

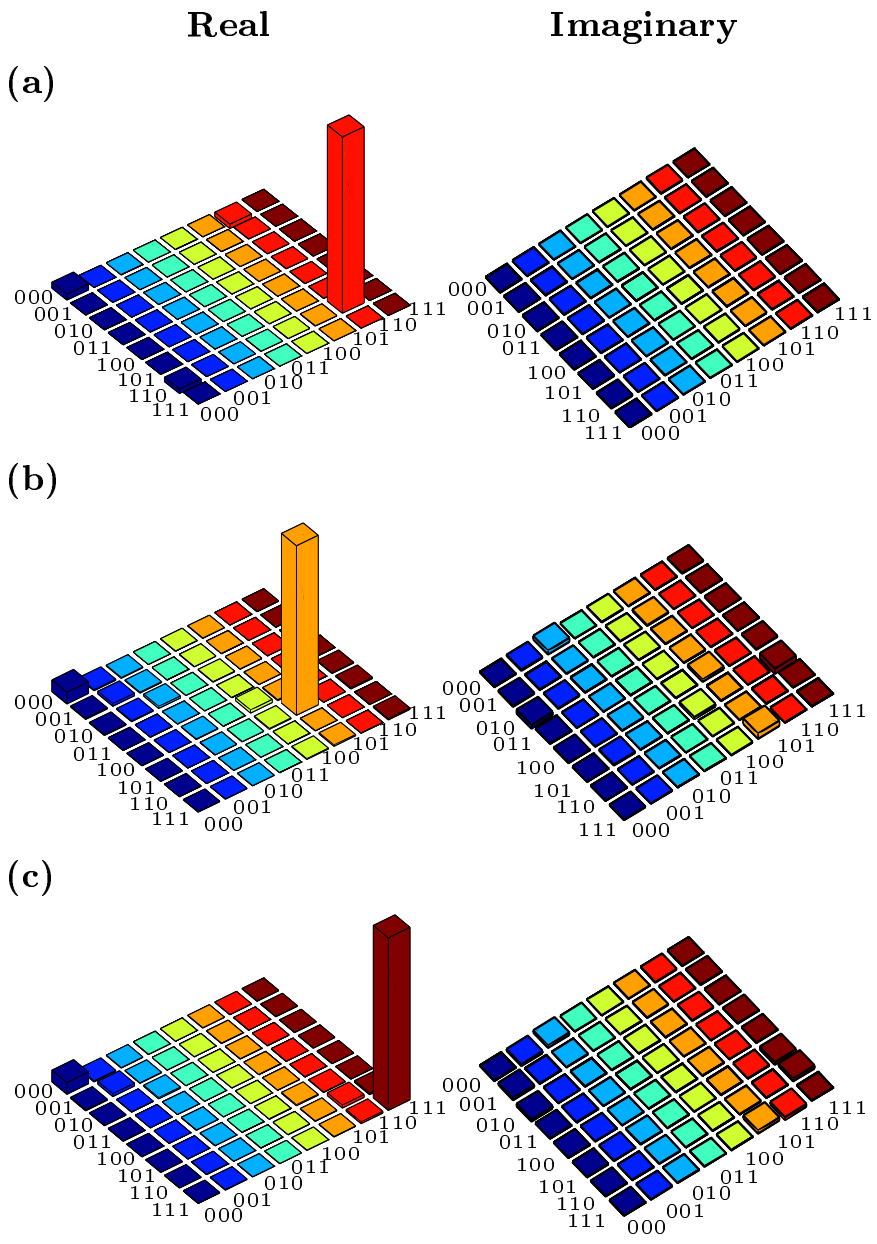


Figure 4.3: Real(left) and imaginary (right) parts of the experimental tomographs of (a)$|110\rangle$ state prepared with fidelity 0.97. (b) after Fredkin(CSWAP) gate on $|110\rangle$ state state with fidelity 0.96 (c) after Toffoli gate on $|110\rangle$ state with fidelity 0.93.

4.4 Conclusion and Future Scopes

The main aim of this work was to design the pulse sequence for the three qubit unitary matrices and to implement them experimentally on an NMR quantum information processor. The technique has only used hard pulses and delays, one can consider this an on and off switch for RF field such that the RF field is turned on only for few microseconds and then turned off and this process repeats until desired unitary is reached. The optimization time for this algorithm was much less than the existing optimization techniques. This optimization technique has helped to obtain the pulse sequence with very high fidelity. The optimized sequence was time optimal and robust which helped to cope up with the different errors associated in NMR quantum information processing. The pulse sequence was experimentally implemented and the output was obtained with very high experimental fidelity.

This optimization technique can be used for optimizing the pulse sequence for any number of qubits but only at the cost of optimization time. The algorithm can be used for quantum state preparation and some of the other NMR experiments where pulse sequence is required. Algorithm can also be used for some other architecture for quantum computing but the changes should be done according to the system.

Appendix A

MATLAB Codes

```
%Codes for optimization of quantum gates. This is the main file
%and rest are the function associated with this file.
%-------------------------------------------------------------------------%
%%Authors- Amit Devra, Prithviraj Prabhu, Harpreet Singh and Kavita
    Dorai
%-------------------------------------------------------------------------%
tic;
z = [1/2 0;0 -1/2];
y = 0.5*[0 -1i;1i 0];
e = [1 0;0 1];
v1 = 34013.57255;%input('freq of spin 1');
v2 = 43480.0746;%input('freq of spin 2');
v3 = 57297.3935;%input('freq of spin 3');
vrf = 43480.0746;%input('rf freq');
J12 = 34.825*2; %input('J12');
J13 = 23.835*2; %input('J13');
J23 = -64.16*2; %input('J23');
iteration = 1;
B = cell(3,1);%for hamiltonian
B{1} = kron(kron(z,e),e);
B{2} = kron(kron(e,z),e);
B{3} = kron(kron(e,e),z);
CNOT = [1 0 0 0;0 1 0 0;0 0 0 1;0 0 1 0];
%Hamiltonian
H = 2*pi*((v1-vrf)*B{1}+(v2-vrf)*B{2}+(v3-vrf)*B{3}+J12*B{1}*B{2}
```

```
+J13*B{1}*B{3}+J23*B{2}*B{3});
%A = kron(kron(p,e),e)+kron(kron(e,p),e)+kron(kron(e,e),p);
%kronecker of phi

U=[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 0
   0 0 0 0 0 0 0 1];
%U = expm(-1i*pi/(2)*kron(kron(eye(2),y),eye(2)));
%U = kron(CNOT,eye(2));

nameOfFile = 'Controlled.txt';
iterations = 6;
flagFitness = 6;
%rows(a)
%-------------------------------------------------------------------------%
diary(nameOfFile);
diary on;

for iteration = 1:iterations
   b = 4; %columns
   a = 15;
   %minRows = 16;
   maxRows = 1;
   FinalCells= cell(maxRows,1);
   FinalFitCells=zeros(maxRows,1);
   maxgen=300;
   thresholdFitness = 0.97;
   ShowAllFitMat=zeros(maxgen,maxRows);
   %for a = minRows:maxRows
   n = 1000;
   Fitness=zeros(n,1);
   k = 1;
   TempArray=cell(maxgen,1);
```

```
%contains the ax4 matrices with highest fitness in each
%generation
TempFitArray=zeros(maxgen,1);
%contains the fitness values coresponding to the above matrices
I = cell(n,1); %initial population array
J = cell(n,1); %temp population array
for k = 1:n
   for q = 1:a
      for w = 1:b
         if w==1
            I{k}(q,w)= 1*10^(-6)+rand(1)*(39*10^(-6)-1*10^(-6));

            %random no. from 1to39micro secs.; for theta
         elseif w==2
            I{k}(q,w)=randi(2)-1;
         elseif w==3
            I{k}(q,w)=randi([0,180],1); %phi column
         elseif w==4
            I{k}(q,w)=rand(1)*(0.005); %delay column
         %elseif w==5
          %  I{k}(q,w)= rand(1)*(0.005);%for delays 0.0100 is max for
              delay
         end
      end
   end
end
gen = 1;
while gen<maxgen+1
   IntermArray=cell(9,1);
   IntermFitArray=zeros(9,1);

   r = rand(1);
   if (gen>maxgen/10 &&r<0.5*gen/maxgen)%mutation of all numbers for
       1/100 of the population
      local = randi(n,round(n/100),1);
      for p=1:n/100
         for q=1:a
            for w=1:b
```

```
            if w==1
               I{local(p)}(q,w)=
                   1*10^(-6)+rand(1)*(39*10^(-6)-1*10^(-6));
            elseif w==2
               I{local(p)}(q,w)=randi(2)-1;
            elseif w==3
               I{local(p)}(q,w)=randi([0,180],1);
            elseif w==4
               I{local(p)}(q,w)=rand(1)*(0.005);
            %elseif w==5
             %  I{local(p)}(q,w)= rand(1)*(0.005);%for delays
                 0.0100 is max for delay
            end
         end
      end
   end
   %calculate fitness
   [Fitness,x8]=fitness(a,n,I,U,flagFitness);
   [val,idx] = max(Fitness);

   if val>thresholdFitness
      disp('Done Done Done');
      disp(val);
      disp(I{idx});
      break;
   end
   IntermFitArray(1)=val;
   IntermArray{1}=I{idx};
end

%cross1
lc=1;
while lc<n
   if gen~=1
      [Chosen1,Chosen2] = luckChoose(Fitness);
      [J{lc},J{lc+1}] = crossover(4,a,I{Chosen1},I{Chosen2},1);
   else
      [J{lc}, J{lc+1}]=crossover(4,a,I{lc},I{lc+1},1);
```

```
      end
      lc = lc+2;
   end
   I = J;
   [Fitness,x8]=fitness(a,n,I,U,flagFitness);

   [val,idx] = max(Fitness);
   if val>thresholdFitness
      disp('DONE DONE DONE');
      disp(val);
      disp(I{idx});
         break;
   end
   IntermFitArray(2)=val;
   IntermArray{2}=I{idx};

   %cross2 %%added this
%    lc=1;
%    while lc<n
%       [Chosen1,Chosen2] = luckChoose(Fitness);
%       [J{lc}, J{lc+1}]=crossover(3,a,I{Chosen1},I{Chosen2},1);
%       %lc=lc+2
%    end
%    I = J;
%    [Fitness,x8]=fitness(a,n,I,U,flagFitness);
%
%    [val,idx]=max(Fitness);
%    if val>thresholdFitness
%       disp('DONE DONE DONE');
%       disp(val);
%       disp(I{idx});
%
%    end
%    IntermFitArray(3)=val;
%    IntermArray{3}=I{idx};

   %flip1
   I=Flip(I,a,n,Fitness);
```

```
    [Fitness,x8]=fitness(a,n,I,U,flagFitness);
    [val,idx]=max(Fitness);

    if val>thresholdFitness
       disp('DONE DONE DONE');
       disp(val);
       disp(I{idx});
          break;
    end
    IntermFitArray(4)=val;
    IntermArray{4}=I{idx};

%     %cross3
%     lc=1;
%     while lc<n
%        [Chosen1,Chosen2] = luckChoose(Fitness);
%        [J{lc}, J{lc+1}]=crossover(3,a,I{Chosen1},I{Chosen2},1);
%        lc=lc+2;
%     end
%     I=J;
%     [Fitness,x8]=fitness(a,n,I,U,flagFitness);
%     [val,idx]=max(Fitness);
%     if val>thresholdFitness
%        disp('DONE DONE DONE');
%        disp(val);
%        disp(I{idx});
%     end
%
%     IntermFitArray(5)=val;
%     IntermArray{5}=I{idx};
%
%     %flip 2
%     I=Flip(I,a,n,Fitness);
%
%     [Fitness,x8]=fitness(a,n,I,U,flagFitness);
%     [val,idx]=max(Fitness);
%     if val>thresholdFitness
%        disp('DONE DONE DONE');
```

```
%          disp(val);
%          disp(I{idx});
%       end
%       IntermFitArray(6)=val;
%       IntermArray{6}=I{idx};

%       %cross4
%
%       lc=1;
%       while lc<n
%          [Chosen1,Chosen2] = luckChoose(Fitness);
%          [J{lc}, J{lc+1}]=crossover(2,a,I{Chosen1},I{Chosen2},1);
%          lc=lc+2;
%       end
%       I=J;
%       [Fitness,x8]=fitness(a,n,I,U,flagFitness);
%       [val,idx]=max(Fitness);
%       if val>thresholdFitness
%          disp('DONE DONE DONE');
%          disp(val);
%          disp(I{idx});
%       end
%       IntermFitArray(7)=val;
%       IntermArray{7}=I{idx};
%
%       %cross5
%
%       lc=1;
%       while lc<n
%          [Chosen1,Chosen2] = luckChoose(Fitness);
%          [J{lc}, J{lc+1}]=crossover(1,a,I{Chosen1},I{Chosen2},1);
%          lc=lc+2;
%       end
%       I=J;
%       [Fitness,x8]=fitness(a,n,I,U,flagFitness);
%       [val,idx]=max(Fitness);
%
%       if val>thresholdFitness
```

```
%          disp('DONE DONE DONE');
%          disp(val);
%          disp(I{idx});
%        end
%        IntermFitArray(8)=val;
%        IntermArray{8}=I{idx};
%
      % second mutation
      %%[I]=secondMutation(Fitness,n,a,b,I);
      %%[Fitness,x8]=fitness(a,n,I,U,flagFitness);
      %%[val,idx]=max(Fitness);

      if val>thresholdFitness
         disp('DONE DONE DONE');
         disp(val);
         disp(I{idx});
            break;
      end
      IntermFitArray(9)=val;
      IntermArray{9}=I{idx};

      [val,idx]=max(IntermFitArray);
      TempFitArray(gen)=val;
      TempArray{gen}=IntermArray{idx};
      %       ShowAllFitMat(gen,a)=val;
      disp(val);
      gen=gen+1;
      disp(gen);
   end
   a
   [val,idx] = max(TempFitArray);
   FinalCells{a}=TempArray{idx};
   FinalFitCells(a)=val;
   %end
   [val, idx] = max(FinalFitCells);
   FinalCell=FinalCells{idx};
   [row,col]=size(FinalCell);
   Y = cell(row,1);
```

```
    l=1;
    for j = 1:row
        COSINE=0.5*(cos(FinalCell(j,3)*pi/180));
        SINE=0.5*((-1)^FinalCell(j,2))*1i*sin(FinalCell(j,3)*pi/180);
        CS=COSINE-SINE;
        CSM=COSINE+SINE;
        %Y{l} =
            expm(((-1)^FinalCell(j,1))*1i*FinalCell(j,2)*pi/180*(kron(kron([0
            CS;CSM 0],e),e)+kron(kron(e,[0 CS;CSM
            0]),e)+kron(kron(e,e),[0 CS;CSM
            0])))*expm((-1)*1i*H*FinalCell(j,5));
        %l = l+1;
        Y{l} = expm(-1i*(H+(pi/(26*10^(-6)))*(kron(kron([0 CS;CSM
            0],e),e)+kron(kron(e,[0 CS;CSM 0]),e)+kron(kron(e,e),[0
            CS;CSM 0])))*FinalCell(j,1))*expm((-1)*1i*H*FinalCell(j,4));
        l = l+1;
    end
    j=1;
    C= Y{j};
    for q = 1:row-1
       C = C*Y{j+q};
    end

    disp('Iteration');
    iteration
    val
    FinalCell
    C
    U

%    for a=minRows:maxRows
%       FinalCells{a}
%       FinalFitCells(a)
%    end

end

diary off;
```

```
MsgBox=msgbox('Finished');
toc;

%%Fitness Function

function [fitnessOfPop,eightxeight]= fitness(a,n,I,U,flag)
%a = 5;
Y = cell(n*a,1);%cell for decomposed matrix(8*8)
Y = decompose(Y,a,n,I);
z = [1/2 0;0 -1/2];
e = [1 0;0 1];
v1 = 34013.57255;%input('freq of spin 1');
v2 = 43480.0746;%input('freq of spin 2');
v3 = 57297.3935;%input('freq of spin 3');
vrf = 43478.655;%input('rf freq');
J12 = 34.825*2; %input('J12');
J13 = 23.835*2; %input('J13');
J23 = -64.16*2; %input('J23');
B = cell(3,1);%for hamiltonian
B{1} = kron(kron(z,e),e);
B{2} = kron(kron(e,z),e);
B{3} = kron(kron(e,e),z);
H = 2*pi*((v1-vrf)*B{1}+(v2-vrf)*B{2}+(v3-vrf)*B{3}+J12*B{1}*B{2}
+J13*B{1}*B{3}+J23*B{2}*B{3});
C = cell(n,1); %product matrix
C = product(C,Y,a,n);
f1 = zeros(n,1);
D = cell(n,1);

if flag==1
    for v = 1:n
        D{v} = ctranspose(C{v});
    end
    for g = 1:n
        f1(g) = 1/trace(abs(D{g}*U-ctranspose(U)*U));
    end
%original fidelity function
elseif flag==6 %currently using
```

```
    C1=zeros(1,n);
    for loopc=1:n
        C1(loopc) = (abs(trace(ctranspose(C{loopc})*U)/8)); % the "abs"
            gets rid of the global phase
        f1(loopc) = C1(loopc);
    end
elseif flag==7
    C1=zeros(1,n);
    for loopc=1:n
        C1(loopc)=real(trace(ctranspose(C{loopc})*U)/8); % the "abs"
            gets rid of the global phase
        f1(loopc) = C1(loopc);
    end
end
[val,idx]=max(f1);
eightxeight=C{idx};
fitnessOfPop=f1;

%%Decompose Function
function decomposition = decompose(Y,a,n,I)
z = [1/2 0;0 -1/2];
e = [1 0;0 1];
v1 = 34013.57255;%input('freq of spin 1');
v2 = 43480.0746;%input('freq of spin 2');
v3 = 57297.3935;%input('freq of spin 3');
vrf = 43478.655;%input('rf freq');
J12 = 34.825*2; %input('J12');
J13 = 23.835*2; %input('J13');
J23 = -64.16*2; %input('J23');
B = cell(3,1);%for hamiltonian
B{1} = kron(kron(z,e),e);
B{2} = kron(kron(e,z),e);
B{3} = kron(kron(e,e),z);
H = 2*pi*((v1-vrf)*B{1}+(v2-vrf)*B{2}+(v3-vrf)*B{3}+J12*B{1}*B{2}+
J13*B{1}*B{3}+J23*B{2}*B{3});
%decompose into a 8x8 Matrix
l = 1;
for k = 1:n
```

```
    for j = 1:a
      COSINE=0.5*(cos(I{k}(j,3)*pi/180));
      SINE=0.5*((-1)^I{k}(j,2))*1i*sin(I{k}(j,3)*pi/180);
      CS=COSINE-SINE;
      CSM=COSINE+SINE;
      Y{l} = expm(-1i*(H+(pi/(26*10^(-6)))*(kron(kron([0 CS;CSM
          0],e),e)+kron(kron(e,[0 CS;CSM 0]),e)+kron(kron(e,e),[0 CS;CSM
          0])))*I{k}(j,1))*expm((-1)*1i*H*I{k}(j,4));
      l = l+1;
    end
end
decomposition=Y;

%%Crossover Function
function[X,Y] = crossover(n,a,X,Y,flag)
b=4; %number of columns
f1 = randi(b,1);%upper bound
g1 = randi(b,1); %lower bound
% if (g1>f1)
%     temp=f1;
%     f1=g1;
%     g1=temp;
% end
f = randi(a,1);%upper bound
g = randi(a,1); %lower bound
% if (g>f)
%     temp=f;
%     f=g;
%     g=temp;
% end

%cross

if (g1>f1)||(g>f)
   if (g1>f1)&&(g>f)
      for l=g1:b
         for j=g:a
            [X(j,l),Y(j,l)] = deal(Y(j,l),X(j,l));
```

```
            end
            for j=1:f
               [X(j,l),Y(j,l)] = deal(Y(j,l),X(j,l));
            end
         end
         for l=1:f1
            for j=g:a
               [X(j,l),Y(j,l)] = deal(Y(j,l),X(j,l));
            end
            for j=1:f
               [X(j,l),Y(j,l)] = deal(Y(j,l),X(j,l));
            end
         end
      elseif (g1>f1)&&(g<f)
         for l=g1:b
            for j = g:f
               [X(j,l),Y(j,l)] = deal(Y(j,l),X(j,l));
            end
         end
         for l=1:f1
            for j = g:f
               [X(j,l),Y(j,l)] = deal(Y(j,l),X(j,l));
            end
         end
      elseif (g1<f1)&&(g>f)
         for l = g1:f1
            for j=g:a
               [X(j,l),Y(j,l)] = deal(Y(j,l),X(j,l));
            end
            for j=1:f
               [X(j,l),Y(j,l)] = deal(Y(j,l),X(j,l));
            end
         end
      end
   else
      for l = g1:f1
         for j = g:f
            [X(j,l),Y(j,l)] = deal(Y(j,l),X(j,l));
```

```
      end
   end
end

%%Flip Function
function I = Flip(I,a,n,Fitness)

[temp,sortIndex] = sort(Fitness);
Isorted = I(sortIndex);
if n~=1
    for q=n/2:n
        if a~=1
            temp1=randi(floor(a/2));
            for w=1:temp1
                i=randi(a);
                j=i;
                while j==i
                    j=randi(a);
                    if j~=i
                        break;
                    end
                end
                Isorted{q}([i j],:) = Isorted{q}([j i],:);
            end
        end
    end
else
    if a~=1
        temp1=randi(floor(a/2));
        for w=1:temp1
            i=randi(a);
            j=i;
            while j==i
                j=randi(a);
                if j~=i
                    break;
                end
            end
```

```
            Isorted{1}([i j],:) = Isorted{1}([j i],:);
        end
    end
end
I = Isorted;

%%Product Function
function productfinal=product(C,Y,a,n)
%give the product of different matrices to give a 8x8 final matrix
v = 1;
j = 1;
for j = 1:a:n*a-a+1
    C{v} = Y{j};
    for q = 1:a-1
        C{v} = C{v}*Y{j+q};
    end
    v = v+1;
end
productfinal=C;

%%This is a Local Optimizer named as 'Member Checker' used for checking
    fitness of member having subjectively high fitness. Process each
    member through it to increase fitness.

%%Member Checker
tic;
z=[1/2 0; 0 -1/2];
e=[1 0; 0 1];
y = 0.5*[0 -1i;1i 0];
v1 = 34011.917;%input('freq of spin 1');
v2 = 43479.26035;%input('freq of spin 2');
v3 = 57296.2954;%input('freq of spin 3');
vrf = 43479.26035;%input('rf freq');
J12 = 34.825*2; %input('J12');
J13 = 23.835*2; %input('J13');
J23 = -64.16*2; %input('J23');
```

```
%p = [0
    0.5*(cos(f*pi/180)+i*sin(f*pi/180));0.5*(cos(f*pi/180)-i*sin(f*pi/180))
    0];%phi
%a = 10; %rows
iteration = 1;
B = cell(3,1);%for hamiltonian
B{1} = kron(kron(z,e),e);
B{2} = kron(kron(e,z),e);
B{3} = kron(kron(e,e),z);
CNOT = [1 0 0 0;0 1 0 0;0 0 0 1;0 0 1 0];
%Hamiltonian
H = 2*pi*((v1-vrf)*B{1}+(v2-vrf)*B{2}+(v3-vrf)*B{3}+J12*B{1}*B{2}+
J13*B{1}*B{3}+J23*B{2}*B{3});
% U=  [ 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 0
%    0 0 0 0 0 0 0 1];
 %U = expm(-1i*pi/(2)*kron(kron(eye(2),eye(2)),y));
%U = kron(CNOT,eye(2));
%
    ----------------------------------START-----------------------------------
Q =[3.00000000000000e-05 1    38.1900000000001  0.000277000000000000
3.30000000000000e-05  1    39.2500000000001  6.90000000000000e-05
3.00000000000000e-06  0    59.0200000000001  1.00000000000000e-06
3.90000000000000e-05  0    66.2800000000001  0.000636000000000000
2.90000000000000e-05  1    53.9400000000001  0.000292000000000000
9.00000000000000e-06  1    57.8800000000003  1.90000000000000e-05
3.90000000000000e-05  1    47.1900000000001  0.00175500000000000
1.10000000000000e-05  1    65.8800000000001  1.00000000000000e-06
3.90000000000000e-05  1    63.8400000000002  0.000636000000000000
1.00000000000000e-06  0    157.809999999997  0.000256000000000000
4.00000000000000e-06  1    88.0300000000001  0.000305000000000000
3.90000000000000e-05  1    30.1400000000001  1.10000000000000e-05
1.40000000000000e-05  1    7.44000000000000  8.30000000000000e-05
```

```
2.40000000000000e-05  1    0.620000000000000  0.000657000000000000
1.00000000000000e-06  0    2.50000000000000  0.00174800000000000
6.00000000000000e-06  0    55.0200000000001  6.90000000000000e-05
4.00000000000000e-06  1    47.5500000000001  2.00000000000000e-06
3.70000000000000e-05  1    50.9300000000001  9.60000000000000e-05];


%Q=Q*100;
alpha=0;% if alpha is 0, it just checks fidelity, if alpha is 1, it
    does one iteration of local optimizer. if alpha is 2, it does
    mutiple iterations
if alpha==0
   FinalCell=Q;
   [row,col]=size(FinalCell);
   Y = cell(row,1);
   l=1;
   for j = 1:row
      COSINE=0.5*(cos(FinalCell(j,3)*pi/180));
      SINE=0.5*((-1)^FinalCell(j,2))*1i*sin(FinalCell(j,3)*pi/180);
      CS=COSINE-SINE;
      CSM=COSINE+SINE;
      %Y{l} =
          expm(((-1)^FinalCell(j,1))*1i*FinalCell(j,2)*pi/180*(kron(kron([0
          CS;CSM 0],e),e)+kron(kron(e,[0 CS;CSM 0]),e)+kron(kron(e,e),[0
          CS;CSM 0])))*expm((-1)*1i*H*FinalCell(j,5));
      %l = l+1;
      Y{l} = expm(-1i*(H+(pi/(26*10^(-6)))*(kron(kron([0 CS;CSM
          0],e),e)+kron(kron(e,[0 CS;CSM 0]),e)+kron(kron(e,e),[0 CS;CSM
          0])))*FinalCell(j,1))*expm((-1)*1i*H*FinalCell(j,4));
      l = l+1;
   end

   j=1;
   C= Y{j};
   for q = 1:row-1
      C = C*Y{j+q};
   end
   C
```

```
   Fidelity=abs(trace(ctranspose(C)*U)/8);
   disp('Fitness');
   fprintf('%.12f ', Fidelity);

   %--------------------testing a population of
       matrices------------------
elseif alpha==1
   %create population
   %call fitness 3
   iterations=1;

   for iteration=1:iterations

      maxgen=10;
      n=2000;
      flagFitness=6;
      J = cell(n, 1); %temp pop array
      % the next two variables are used to determine the matrix with the
      % highest fitness among all generations
      Pop=cell(n,1);
      [rows,cols]=size(Q);
      a=rows;
      TempArray=cell(maxgen,1);
      %contains the ax4 matrices with highest fitness in each generation
      TempFitArray=zeros(maxgen,1);
      %contains the fitness values corresponding to the above matrices
      colindex=2;

      %prepare initial population
      for popcount=1:n
         Pop{popcount}=Q;
         for row=1:rows
            if colindex==5
               Pop{popcount}(row,5)=Q(row,5)+rand(1)*0.00001-0.000005
               ;%----------changed this
            else
               Pop{popcount}(row,colindex)=Q(row,colindex)+rand(1)*10-5;
            end
```

```
   end
end
disp('created pop');
gen=1;
[Fitness,x8]=fitness(a,n,Pop,U,flagFitness);
while gen<maxgen+1
   % do cross 4/5
   lc=1;
   while lc<n
      [ First,Second ]= luckChoose(Fitness);
      [J{lc},
         J{lc+1}]=Cross(colindex,rows,Pop{First},Pop{Second},0);
         %Cross(X,rows....) where X is the index of the column
         needing crossing over
      lc=lc+2;
   end
   Pop=J;
   [Fitness,x8]=fitness(a,n,Pop,U,flagFitness);
   [val,idx]=max(Fitness);
   TempFitArray(gen)=val;
   TempArray{gen}=Pop{idx};

   disp(val);
   disp(gen);
   gen=gen+1;
end

[val ,idx]=max(TempFitArray);
Fitval=val;
Fit=TempArray{idx};

Fit
FinalCell=Fit;
[row,col]=size(FinalCell);
Y = cell(row,1);
l=1;
for j = 1:row
   COSINE=0.5*(cos(FinalCell(j,3)*pi/180));
```

```
        SINE=0.5*((-1)^FinalCell(j,2))*1i*sin(FinalCell(j,3)*pi/180);
        CS=COSINE-SINE;
        CSM=COSINE+SINE;
        %Y{l} = expm(((-1)^FinalCell(j,1))*1i*FinalCell(j,2)
        %*pi/180*(kron(kron([0 CSM;CS 0],e),e)+kron(kron(e,[0 CSM;CS
            0]),e)+kron(kron(e,e),[0 CSM;CS
            0])))*expm((-1)*1i*H*FinalCell(j,5));
        %l = l+1;
        Y{l} = expm(-1i*(H+(pi/(26*10^(-6)))*(kron(kron([0 CS;CSM
            0],e),e)+kron(kron(e,[0 CS;CSM 0]),e)+kron(kron(e,e),[0
            CS;CSM 0])))*FinalCell(j,1))*expm((-1)*1i*H*FinalCell(j,4));
        l = l+1;
      end


      j=1;
      C= Y{j};
      for q = 1:row-1
        C = C*Y{j+q};
      end
      C
      Fidelity=(abs(trace(ctranspose(C)*U)/8))^(2);
      disp('Fitness');
      fprintf('%.12f ', Fidelity);
    end
    MsgBox=msgbox('Finished');
  elseif alpha==2
    iterations=1;
    for iteration=1:iterations
      QQ=Q;
      ite=1;
      colindexArray=[3 1 3 4];
      indexCol=randi(length(colindexArray)-1);
      colIndexFailure=0;
      Fidelity0=0;
      Fidelity1=0;
      col4start=0.000005;
      col3start=10;
```

```
col1start=0.000003;
col4startby2=col4start/2;
col3startby2=col3start/2;
col1startby2=col1start/2;
FinalCell=zeros(12,5);
while(1)
   if (ite~=1 )&&(colIndexFailure==0)
      QQ=FinalCell;
   end
   maxgen=8;
   n=200;
   flagFitness=6;
   J = cell(n, 1); %temp pop array
   % the next two variables are used to determine the matrix with
       the
   % highest fitness among all generations
   Pop=cell(n,1);
   [rows,cols]=size(QQ);
   a=rows;
   TempArray=cell(maxgen,1);
   %contains the ax4 matrices with highest fitness in each
       generation
   TempFitArray=zeros(maxgen,1);
   %contains the fitness values corresponding to the above matrices
   colindex=colindexArray(indexCol)
   %prepare initial population
   for popcount=1:n
      Pop{popcount}=QQ;
      for row=1:rows
         if colindex==4
            Pop{popcount}(row,4)=QQ(row,4)+rand(1)*col4start
            -col4startby2;%----------changed this
         elseif colindex==3
            Pop{popcount}(row,colindex)=QQ(row,colindex)
            +rand(1)*col3start-col3startby2;%--------alpha 2!!
         elseif colindex==1
            Pop{popcount}(row,colindex)=QQ(row,colindex)
            +rand(1)*col1start-col1startby2;%--------alpha 2!!
```

```
      end
   end
end
disp('created pop');
gen=1;
[Fitness,x8]=fitness(a,n,Pop,U,flagFitness);
while gen<maxgen+1
   % do cross 4/5
   lc=1;
   while lc<n
      [ First,Second ]= luckChoose(Fitness);
      [J{lc},
         J{lc+1}]=Cross(colindex,rows,Pop{First},Pop{Second},0);
         %Cross(X,rows....) where X is the index of the column
         needing crossing over
      lc=lc+2;
   end
   Pop=J;
   [Fitness,x8]=fitness(a,n,Pop,U,flagFitness);
   [val,idx]=max(Fitness);
   TempFitArray(gen)=val;
   TempArray{gen}=Pop{idx};

   %            disp(val);
   %            disp(gen);
   gen=gen+1;
end

[val ,idx]=max(TempFitArray);
Fitval=val;
FinalCell=TempArray{idx};


FinalCell
[row,col]=size(FinalCell);
Y = cell(row,1);
l=1;
for j = 1:row
```

```
   COSINE=0.5*(cos(FinalCell(j,3)*pi/180));
   SINE=0.5*((-1)^FinalCell(j,2))*1i*sin(FinalCell(j,3)*pi/180);
   CS=COSINE-SINE;
   CSM=COSINE+SINE;
   %Y{l} =
       expm(((-1)^FinalCell(j,1))*1i*FinalCell(j,2)*pi/180*(kron(kron([0
       CSM;CS 0],e),e)+kron(kron(e,[0 CSM;CS
       0]),e)+kron(kron(e,e),[0 CSM;CS
       0])))*expm((-1)*1i*H*FinalCell(j,5));
   %l = l+1;
   Y{l} = expm(-1i*(H+(pi/(26*10^(-6)))*(kron(kron([0 CS;CSM
       0],e),e)+kron(kron(e,[0 CS;CSM 0]),e)+kron(kron(e,e),
   [0
       CS;CSM0])))*FinalCell(j,1))*expm((-1)*1i*H*FinalCell(j,4));
   l = l+1;
end

j=1;
C= Y{j};
for q = 1:row-1
   C = C*Y{j+q};
end
C
Fidelity=abs(trace(ctranspose(C)*U)/8);
disp('Fitness');
fprintf('%.12f ', Fidelity);
if (ite~=1 )&&(colIndexFailure==0)
   Fidelity1=Fidelity0;
end
Fidelity0=Fidelity; %Fidelity0 must be greater than Fidelity1;
if Fidelity1>=Fidelity0
   colIndexFailure=colIndexFailure+1;
   if colIndexFailure==length(colindexArray)
      disp('No better solutions found. Sorry. Check if you
          want.');
      col4start=col4start/1.5;
      col3start=col3start/1.5;
      col1start=col1start/1.5;
```

```
            col4startby2=col4start/2;
            col3startby2=col3start/2;
            col1startby2=col1start/2;
            colIndexFailure=0;
            %            break;
         end
      else
         colIndexFailure=0;
      end

      indexCol=indexCol+1;
      if indexCol>length(colindexArray)
         indexCol=1;
      end
      ite=ite+1;
      colIndexFailure
      col3start
   end
   MsgBox=msgbox('Finished');
end
elseif alpha==3
   ite=1;
   colindexArray=[3 1 4];
   indexCol=randi(length(colindexArray)-1);
   colIndexFailure=0;
   Fidelity0=0;
   Fidelity1=0;

   while(1)
      if (ite~=1 )&&(colIndexFailure==0)
         Q=FinalCell;
      end
      n=100;
      flagFitness=6;
      maxgen=10;
      J = cell(n, 1); %temp pop array
      % the next two variables are used to determine the matrix with the
      % highest fitness among all generations
```

```
Pop=cell(n,1);
[rows,cols]=size(Q);
a=rows;
TempArray=cell(maxgen,1);
%contains the ax4 matrices with highest fitness in each generation
TempFitArray=zeros(maxgen,1);
%contains the fitness values corresponding to the above matrices
colindex=colindexArray(indexCol)
%prepare initial population
for popcount=1:n
   Pop{popcount}=Q;
   for row=1:rows
      if colindex==4||1
         Pop{popcount}(row,colindex)=Q(row,colindex)
         +randi(3)*0.000001-0.000002;
         %you must always give randi(odd_number)-(odd_number+1).
            because, for example, if you take randi(2)-1, the only
            possible values are 0,1. but if you take rand(3)-2,
            you get -1,0,1.
         maxgen=25;
      else
         Pop{popcount}(row,colindex)=Q(row,colindex)+randi(501)*0.01-2.51;
         maxgen=10;
      end
   end
end
disp('created pop');
gen=1;
[Fitness,x8]=fitness(a,n,Pop,U,flagFitness);
while gen<maxgen+1
   % do cross 4/5
   lc=1;
   while lc<n
      [ First,Second ]= luckChoose(Fitness);
      [J{lc},
         J{lc+1}]=Cross(colindex,rows,Pop{First},Pop{Second},0);
         %Cross(X,rows....) where X is the index of the column
         needing crossing over
```

```
      lc=lc+2;
   end
   Pop=J;
   [Fitness,x8]=fitness(a,n,Pop,U,flagFitness);
   [val,idx]=max(Fitness);
   TempFitArray(gen)=val;
   TempArray{gen}=Pop{idx};

   disp(val);
   disp(gen);
   gen=gen+1;
end

[val ,idx]=max(TempFitArray);
Fitval=val;
Fit=TempArray{idx};

Fit
FinalCell=Fit;
[row,col]=size(FinalCell);
Y = cell(row,1);
l=1;
for j = 1:row
   COSINE=0.5*(cos(FinalCell(j,3)*pi/180));
   SINE=0.5*((-1)^FinalCell(j,2))*1i*sin(FinalCell(j,3)*pi/180);
   CS=COSINE-SINE;
   CSM=COSINE+SINE;
   %Y{l} =
       expm(((-1)^FinalCell(j,1))*1i*FinalCell(j,2)*pi/180*(kron(kron([0
       CSM;CS 0],e),e)+kron(kron(e,[0 CSM;CS
       0]),e)+kron(kron(e,e),[0 CSM;CS
       0])))*expm((-1)*1i*H*FinalCell(j,5));
   %l = l+1;
   Y{l} = expm(-1i*(H+(pi/(26*10^(-6)))*(kron(kron([0 CS;CSM
       0],e),e)+kron(kron(e,[0 CS;CSM 0]),e)+kron(kron(e,e),[0
       CS;CSM 0])))*FinalCell(j,1))*expm((-1)*1i*H*FinalCell(j,4));
   l = l+1;
end
```

```
      j=1;
      C= Y{j};
      for q = 1:row-1
         C = C*Y{j+q};
      end
      C
      Fidelity=abs(trace(ctranspose(C)*U)/8);
      disp('Fitness');
      fprintf('%.12f ', Fidelity);
      if (ite~=1 )&&(colIndexFailure==0)
         Fidelity1=Fidelity0;
      end
      Fidelity0=Fidelity; %Fidelity0 must be greater than Fidelity1;
      if Fidelity1>=Fidelity0
         colIndexFailure=colIndexFailure+1;
         if colIndexFailure==length(colindexArray)
            disp('No better solutions found. Sorry. Check if you want.');
            break;
         end
      else
         colIndexFailure=0;
      end

      indexCol=indexCol+1;
      if indexCol>length(colindexArray)
         indexCol=1;
      end
      ite=ite+1;
      colIndexFailure
   end
   MsgBox=msgbox('Finished');
end
toc;
```

Bibliography

[AA09] Anil Kumar Ashok Ajoy, *A new hierarchical genetic algorithm approach to determine pulse sequences in nmr*, arXiv:0911.5465v2 [quant-ph] (2009).

[AJ01] Jonathan A. Jones, *Quantum computing and nuclear magnetic resonance*, PhysChemComm **4** (2001), 49–56.

[Bac94] T. Back, *Selective pressure in evolutionary algorithms: a characterization of selection mechanisms*, IEEE (1994).

[DGCH97] Amr F. Fahmy David G. Cory and Timothy F. Havel, *Ensemble quantum computing by nmr spectroscopy*, Proc Natl Acad Sci (1997).

[DiV00] David P. DiVincenzo, *The physical implementation of quantum computation*, arXiv:quant-ph/0002077 (2000).

[D.M95] K.M.Ho D.M.Deaven, *Molecular geometry optimization with a genetic algorithm*, Physical Review Letters,Vol. 75 (1995).

[GBM16] V. S. Anjusha Gaurav Bhole and T. S. Mahesh, *Steering quantum dynamics via bang-bang control: Implementing optimal fixed-point quantum search algorithm*, PHYSICAL REVIEW A 93, 042339 (2016) (2016).

[G.S06] D.S.Linden G.S.Hornby, A.Globus, *Automated antenna design with evolutionary algorithms*, IEEE (2006).

[Hol92] John H. Holland, *Adaptation in natural and artificial systems*, Jan 1992.

[HS16] Kavita Dorai Harpreet Singh, Arvind, *Constructing valid density matrices on an nmr quantum information processor via maximum likelihood estimation*, Physics Letters A Volume 380, Issue 38 (2016).

[IO07] Tito Bonagamba Eduardo Azevedo Jair C. C. Freitas Ivan Oliveira, Roberto Sarthour Jr., *Nmr quantum information processing*, 1st ed., Elsevier Science, May 2007.

[JAS96] David P. DiVincenzo John A. Smolin, *Five two-bit quantum gates are sufficient to implement the quantum fredkin gate*, J Magn Reson. 2005 (1996).

[KJ13] Mohammed Madiafi Khalid Jebari, *Selection methods for genetic algorithms*, Int. J. Emerg. Sci., 3(4), 333-344 (2013).

[MAT15] MATLAB, *Version 8.5.0 (r2015a)*, MathWorks Inc., Natick, Massachusetts, 2015.

[Mit96] Melanie Mitchell, *An introduction to genetic algorithms*, MIT Press. ISBN 9780585030944 (1996).

[MK12] V. S. Manu and Anil Kumar, *Singlet-state creation and universal quantum computation in nmr using a genetic algorithm*, PHYSICAL REVIEW A 86, 022324 (2012) (2012).

[Moo65] Gordon E. Moore, *Cramming more components onto integrated circuits*, Electronics, Volume 38, Number 8 (1965).

[NC11] Michael A. Nielsen and Isaac L. Chuang, *Quantum computation and quantum information: 10th anniversary edition*, 10th ed., Cambridge University Press, New York, NY, USA, 2011.

[NK05] C. Kehlet T. Schulte-Herbrggen S. J.Glaser N. Khaneja, T. Reiss, *Optimal control of coupled spin dynamics: design of {NMR} pulse sequences by gradient ascent algorithms*, Journal of Magnetic Resonance **172** (2005), no. 2, 296 – 305.

[S.M08] A.M.Noor S.Mashohor, K.Samsudin, *Evaluation of genetic algorithm based solar tracking system for photovoltaic panels*, IEEE (2008).

[T.G13] A.F.Stappen T.Geijtenbeek, M.V.Panne, *Flexible muscle-based locomotion for bipedal creatures*, ACM Transactions on Graphics, Vol. 32 (2013).

[XF02] Shi Mingjun Zhou Xianyi-Han Rongdian Wu Jihui Xue Fei, Du Jiangfeng, *Realization of fredkin gate by three transition pulses in nmr quantum information processor*, Chin. Phys. Lett. 2002, vol.19 No.8 p1048 (2002).