On the Evolutionary Design of Quantum Circuits

by

Tim Reid

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2005

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.
I understand that my thesis may be made electronically available to the public.

Abstract

The goal of this work is to understand the application of the evolutionary programming approach to the problem of quantum circuit design. This problem is motivated by the following observations:

- In order to keep up with the seemingly insatiable demand for computing power our computing devices will continue to shrink, all the way down to the atomic scale, at which point they become quantum mechanical systems. In fact, this event, known as Moore's Horizon, is likely to occur in less than 25 years.
- The recent discovery of several quantum algorithms which can solve some interesting problems more efficiently than any known classical algorithm.
- While we are not yet certain that quantum computers will ever be practical to build, there do now exist the first few astonishing experimental devices capable of briefly manipulating small quantities of quantum information. The programming of these devices is already a nontrivial problem, and as these devices and their algorithms become more complicated this problem will quickly become a significant challenge.

The Evolutionary Programming (EP) approach to problem solving seeks to mimic the processes of evolutionary biology which have resulted in the awesome complexity of living systems, almost all of which are well beyond our current analysis and engineering capabilities. This approach is motivated by the highly successful application of Koza's Genetic Programming (GP) approach to a variety of circuit design problems, and specifically the preliminary reports by Williams and Gray and also Rubinstein who applied GP to quantum circuit design.

Accompanying this work is software for evolutionary quantum circuit design which incorporates several advances over previous approaches, including:

- A formal language for describing parallel quantum circuits out of an arbitary elementary gate set, including gates with one or more parameters.
- A fitness assessment procedure that measures both average case fidelity with a respect for global phase equivalences, and implementation cost.

- A Memetic Programming (MP) based reproductive strategy that uses a combination of global genetic and local memetic searches to effectively search through diverse circuit topologies and optimize the parameterized gates they contain.

Several benchmark experiments are performed on small problems which support the conclusion that Evolutionary Programming is a viable approach to quantum circuit design and that further experiments utilizing more computational resources and more problem insight can be expected to yield many new and interesting quantum circuits.

Contents

Chapter 1

Introduction

The goal of this work is to understand the application of the evolutionary programming approach to problem of quantum circuit design.

The first chapter provides a brief overview of quantum computing, including a short introduction to quantum mechanics and computer science. The section concludes with a discussion of the particular quantum circuit formalism employed by this work, and a formal statement of the quantum circuit design problem that is the focus of this work.

The second chapter provides a brief overview of evolutionary programming, including a short introduction to evolution, how it can be seen as a search algorithm, and a history of the main attempts to apply evolutionary concepts to problem solving.

Next is a discussion of the application of genetic programming to the specific problem of circuit design, known as evolutionary circuit design. This section first examines the highly successful preliminary work by Koza and others who have evolved many interesting analog and digital electronic circuits. We then consider the very recent successful applications, first by Williams and Gray, of evolutionary circuit design to quantum circuits. These clear and positive reports are the motivation for this work.

Building on the general framework developed by Koza, and guided by the work of Williams and Gray, the next chapter provides a thorough overview of a proposed approach to evolutionary quantum circuit design which is implemented by the accompanying software. The software is designed to be fast, flexible, and easy to use by a wide audience.

The next chapter contains a report on several experiments that were performed in order to test the effectiveness of the proposed approach, and to provide benchmarks for comparison with future approaches. Each experiment yielded a successful solution to a small but still significant quantum circuit design problem.

The final chapter reflects upon the experimental results, and concludes that the evolutionary approach to quantum circuit design is viable. This work ends with a discussion of some directions for future research, including the need for larger scale experiments.

Chapter 2

What is Quantum Computing?

> *The theory of computation has traditionally been studied almost entirely in the abstract, as a topic in pure mathematics. This is to miss the point of it. Computers are physical objects, and computations are physical processes. What computers can or cannot compute is determined by the laws of physics alone, and not by pure mathematics.*
>
> **David Deutsch [12]**

Quantum computing is the product of two of the most significant advances in the history of science: the theory of quantum mechanics, which describes the universe at the smallest scale with great accuracy, and the theory of computation, which has resulted in digital computers with exponentially increasing power and many efficient algorithms that are capable of solving a wide range of interesting problems.

2.1 The Quantum Age

Anybody who is not shocked by quantum theory has not understood it.

Niels Bohr [41]

The 19th century saw many great advances in science and technology that transformed our understanding of the universe. For a brief period it seemed that all of the natural world would soon be explained by a compact, intuitive, and deterministic theory based primarily on Newtonian mechanics and Maxwell's Electromagnetism, which is now known retrospectively as classical physics. But at the turn of the 20th century there were three easily reproducible experiments which were to defy the greatest efforts of those who would attempt to explain them with the classical physics. The attempt to understand these anomalous experiments lead directly to the shocking discovery that energy does not exist in arbitrary amounts, but only in whole multiples of some vanishingly small but finite quantity, known as a quanta.

2.1.1 Quantum Mechanics

Soon after the momentous discovery of the quanta the work of Schrödinger, Heisenberg, and Dirac yields a new and more correct theory of physics which entailed an entirely new and strange description of particles and their behavior. This theory, now known as Quantum Mechanics, uses the quanta to describe the universe at the smallest scale with astonishing accuracy, and yet can be broadly characterized with just three relatively simple postulates.

The first postulate, known as the state space postulate, states that any quantum mechanical system S is mathematically described by a Hilbert space. Each distinct state of the system is represented by a unit length vector commonly denoted ψ. If a system is composed of two distinct independant systems, then the state space of this composite system is obtained by applying the tensor product to the component state spaces.

$$S = S_1 + S_2 \Longleftrightarrow H = H_1 \otimes H_2 \tag{2.1}$$

The second postulate, known as the time evolution postulate, states that for any physically possible transformation of a closed quantum mechanical system from state ψ_1 to state ψ_2, there is a corresponding unitary operator U, known as the time evolution operator, such that:

$$\psi_2 = U\psi_1 \tag{2.2}$$

The third postulate, known as the measurement postulate, states that for each observable measurement outcome ϕ of a quantum mechanical system there is a corresponding measurement operator M_ϕ, such that if the system is in state ψ then the probability of observing ϕ is given by:

$$\Pr(\phi) = (M_\phi\psi)^\dagger(M_\phi\psi) \tag{2.3}$$

After measurement, the system is in the state ψ_2 which is given by:

$$\psi_2 = \frac{M_\phi\psi}{\sqrt{\Pr(\phi)}} \tag{2.4}$$

Despite the relative simplicity of these postulates they imply a fantastically strange picture of the universe that bares little or no resemblance to everyday experience.

One famous example is the Superposition Principle, which states that while quantum mechanical systems are always observed to be in a single definite state, when they are not observed they can also exist in an indefinite blend of states, known as a superposition. This is the subject of the famous Schrödinger's Cat parable, in which a live cat is placed in a box along with a vial of poison, a radioactive substance, and a device that will release the poison when it detects the radioactive decay of the substance. When the box is open the cat is known with certainty to be alive. However, once the box is sealed, quantum mechanics implies that the time at which the radioactive substance will undergo decay cannot be known with certainty. Therefore we cannot be certain if the poison has been released, and hence the state of the cat is also uncertain. In fact, the cat must be considered to be both alive and dead until the box is opened and the cat is observed. While this parable was originally intended by Schrödinger to dismiss the concept of a superposition as an obvious absurdity, it is now accepted as a valid consequence of quantum mechanics.

The discovery of quantum mechanics sent out a shockwave as many philosophical domains were forced to respond to this strange new understanding of the natural world. Quantum mechanics has been tested with unprecedented accuracy, making it one of the most successful scientific theories in history, but it is not entirely perfect. In particular there as yet remains no way to represent the force of gravity in such a way as to be consistent with the other celebrated theory of physics, general relativity. Attempts to unify these theories have so far been unsuccessful, but it is clear that the question is of the utmost importance if we are to have a more complete and unified theory of physics.

2.2 The Computing Age

> *In natural science, Nature has given us a world and we're just to discover its laws. In computers, we can stuff laws into them and create a world.*
>
> **Alan Kay [42]**

The beginning of the history of computing is lost in ancient history, and is perhaps as old as life itself. Humans probably began by representing numbers with fingers, collections of pebbles, marks pressed into clay, and later symbols written onto paper. The ability to represent numbers immediately motivated the invention of various tools to aid in the performance of arithmetic. One of the earliest surviving examples is the abacus, which is a hand operated calculator that was independently discovered by many ancient civilizations and is still in use today, over 2800 years later.

The fundamental concept of computation is the algorithm, which is an unambiguous procedure for constructing a solution to a certain problem. The word algorithm originates from the name of 9th century Persian mathematician Abu Abdullah Muhammad bin Musa al-Khwarizmi, who published several seminal scientific texts, including *Al-Jabr wa-al-Muqabilah*. This text contains procedures for solving certain linear and quadratic equations that were studied in Europe for over 500 years, and its title serves as the origin of the term algebra.

Over the next millennium the desire to perform ever increasingly complicated calculations resulted in ever increasingly clever mechanical devices. However these devices all performed essentially the same fixed set of calculations which were used to solve essentially the same, albeit larger, arithmetical problems. The first proposal for a general purpose programmable computer was the analytical engine designed by Babbage in 1831. This design used steam to power a complex system of gear driven mechanics which processed programs encoded onto punched cards to generate output on a text printer or graphical curve plotting device. Unfortunately the analytical engine required such mechanical precision that it was too expensive to build at the time. This was an early indication of the main obstacle to large, precise, general purpose computing devices, namely the inevitable minor faults in the components of the computer. These minor faults quickly multiply and compound to become major faults which destroy the accuracy of the output. The only technique for dealing with these faults at the time was to keep the computing device simple, slow, and as precisely manufactured, and therefore expensive, as possible.

2.2.1 The Formalization of the Algorithm

The effort to raise the study of computation to a fully formalized theory begins with a challenge made at the turn of the 20th century by Hilbert to clarify the consistency of arithmetic. In particular, Hilbert's challenge implies an algorithm to determine the validity of a mathematical proof, thereby removing the necessity of fallible human intuition. However, in 1930 Gödel proved the much celebrated Incompleteness Theorem [17], which states that any consistent formal system of sufficient complexity is necessarily incomplete. That is, there are some problems for which there do not exist algorithms. Despite this indication of a fundamental limit, multiple attempts were made to formalize the intuitive notion of an algorithm. In 1936, Turing published a celebrated result [62] which showed that all of the different models of computation, including the Universal Turing Machine (UTM), were in fact completely equivalent. To this day, no fundamentally more powerful but still practical model of computation has been found, and it is a widely held conjecture, known as the Church Turing Thesis, that no such models exist.

The study of the algorithm continued by further classifying them according to their required amount of computational resources, such as storage space or execution time, as a function of their input size. The formalization of computational complexity began in the early 1960's, establishing an important new field of computer science [20] [8] [23]. An algorithm is defined to be efficient if the amount of resources it requires scales no faster than some polynomial in the input size, and inefficient if it grows faster than any polynomial. A computational problem is then defined to be easy if there exists an efficient algorithm to solve it, and if no such algorithm exists then the problem is hard.

A large class of interesting problems, such as database searching and sorting, much of algebra, and many engineering simulations turn out to be easy. In addition, in a conjecture known as the Strong Church Turing Thesis, the emulation of any other algorithmic device is an easy problem for a UTM. Occasionally, the discovery of the first efficient algorithm for a problem can change its status from hard to easy, but there is a large class of interesting problems that appear to be intrinsically hard. The only way we can solve large instances of hard problems is to build massively powerful computers, or to discover a new and more powerful model of computation.

2.2.2 The Digital Computer Revolution

The first and only practical, reliable, scalable, and general purpose computing device is the digital computer. Its invention yielded a sudden availability of ever increasing computing power that is so significant that it serves as the transition point in history between the Industrial Age and the Computing Age. The digital computer was itself made possible by tremendously synergistic breakthroughs in three domains: computability theory, coding theory, and computer engineering.

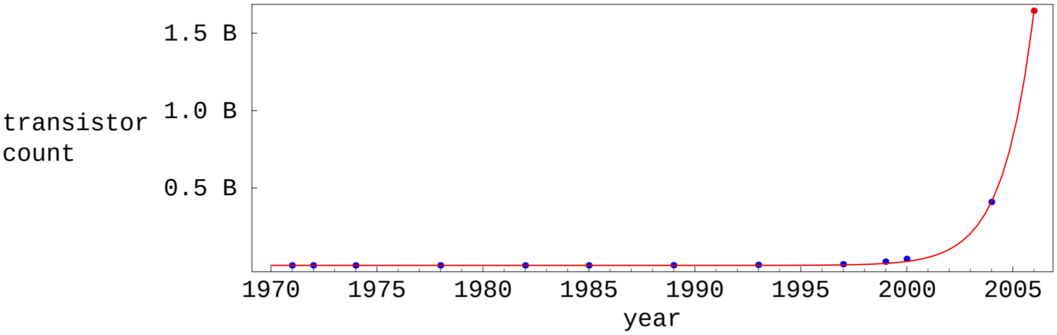
The breakthrough in computability theory was the invention of the digital computer. In 1937 Shannon proposed the Digital Circuit Model [54], which showed how to practically implement the Boolean algebra system using readily available electronic switches. Then in 1946, von Neumann and others developed the von Neumann Architecture [7] which provided a basic framework for the design of scalable and general purpose digital computers using digital circuits. Nearly every computer ever made has followed this design.

The breakthrough in coding theory was the invention of efficient error correcting digital circuits which resulted from the work Shannon [55], Hamming [53], and Huffman [22] circa 1950. These techniques made it possible for the first time to build extremely reliable computing devices out of less reliable parts.

The breakthrough in computer engineering was the invention of the microprocessor. This began with the invention of the transistor by Shockley, Bardeen, and Brattain at Bell Laboratories in 1947. The transistor immediately became an important tool for computer engineering because it could be used as a tiny, reliable, and cost effective digital circuit component. While the transistor did facilitate several early digital computers, the real impact wasn't felt until the introduction of the Intel 4004 digital microprocessor in 1971. This first microprocessor contained more than 2300 transistors on a 3 by 4 millimeter chip, and could be used as the core of a reasonably low cost, reliable, and general purpose digital computer.

2.2.3 Moore's Law

Since the invention of the microprocessor the demand for computing power has been seemingly insatiable, and has motivated unprecedented levels of research and development in the rapidly maturing fields of computer science and computer engineering. As a result, the transistor count of a typical microprocessor is doubling approximately every 18 months, as shown in 2.5. This trend was first observed by Moore in 1965 [40], and his prediction that this trend would continue is known as Moore's Law. The red curve in 2.5 shows the extrapolation of this trend to 2006.



(2.5)

In practical terms, Moore's Law means that the amount digital computing power that can be obtained for a fixed cost is doubling approximately every 18 months. So for as long as this trend continues, we can expect to solve instances of easy problems that are twice as large, and slightly larger instances of hard problems. However, to fully realize this astonishing explosion of computing power requires the design and implementation of ever increasingly complicated hardware and software devices, and current strategies are already struggling to keep up.

One profound obstacle is Moore's Horizon, which refers to the time when transistors will have been shrunk all the way down to the atomic scale. At that point computing devices will enter the domain of quantum mechanics. We cannot yet efficiently engineer complicated devices at that scale, nor is it clear that we will be able to do so anytime soon. In addition, the error inducing noise that atomic scale elements would be subjected to is radically different than its current day large scale counterpart, and therefore the current methods of error correction would no longer work. However, at the current pace Moore's Horizon is less than 25 years away. This naturally leads to the question as to whether computing with quantum mechanical systems will ever be practical, or even advantageous.

2.3 Quantum Computing

In 1980, Benioff [4] expressed the first ideas relating quantum mechanics to computation. Shortly thereafter, Feynman [13] and Manin [39] independently observed that while the simulation of quantum mechanics on a classical computing device was possible, it seemed to necessarily require the representation and manipulation of exponentially large objects, and it follows that no classical computing device can perform this simulation efficiently.

Deutsch was the first to explore the possibilities of computing with quantum mechanical systems which resulted in the quantum analog of the UTM, known as the Universal Quantum Computer (UQC) [12]. Since quantum mechanics can be simulated by classical computers, the UQC and UTM can solve the same set of problems. However, since the simulation of quantum mechanics is believed to be a hard problem for classical computers, this leads to the surprising observation that there are some computations that a UQC can perform efficiently that a UTM cannot. Therefore the Strong Church Turing Thesis would appear to be false. It remains an important open question, perhaps best described as the Strong Church Turing Deutsch Thesis, as to whether or not the UQC can efficiently simulate all physical systems.

Considerable recent research has explored which other kinds of computations, other than physical simulations, that a quantum computer might perform more efficiently than a classical computer. Deutsch gave the first glimpse of the tremendous potential of quantum algorithms by showing that by framing the problem in terms of quantum mechanics, it was possible to obtain global parity information about a binary function by only evaluating the function once, where any classical algorithm must necessarily evaluate the function twice [12]. While this is not a very interesting problem, it marked the beginning of intensive research into the area which has resulted in several algorithms that do solve some interesting problems, including:

- The Quantum Search Algorithm [18] can be expected to find a particular element in an unstructured list of n elements in only $\sqrt{n}$ steps, whereas any classical search is expected to take $\frac{n}{2}$ steps. This algorithm can be applied extremely broadly as an efficiency boosting subroutine.

- The Quantum Fourier Transform (QFT) [43] can perform the quantum analog of the Fourier transformation on a n qubit quantum register in only $n^2 \log(n) \log(\log(n))$ steps, which is exponentially faster than the $n^2 2^n$ many steps of the classical algorithm. While the QFT does not give all of the information provided by the classical version, it can still be applied to a wide range of interesting problems, such as the hidden subgroup problem, order finding in groups, counting, and the discrete log problem.
- Shor's Algorithm [57] uses the QFT as a subroutine to efficiently factor large numbers which remains a hard problem for classical computers. This algorithm is of particular practical interest because it can be used as part of an effective attack on the popular RSA encryption protocol.

Most research in quantum algorithms is expressed in the quantum analog of the classical circuit model first formulated by Deutsch [11], known as quantum circuits. As in the classical case, quantum circuits are equivalent in power to the UQC, but they are generally more intuitive to analyze and construct. However, the design of quantum circuits is not any easier than previously encountered types of circuits. The very first prototype quantum computers are now appearing in research labs. These experimental devices contain on the order of 10 qubits and are only capable of implementing very simple quantum circuits before succumbing to noise. However, even these simple circuits are already non trivial to design.

It would seem that the only way to know if fully fledged quantum computers are practical is to continue to try and build them. The next step will be the invention of the quantum transistor, which would provide a low cost, scalable, and sufficiently reliable physical implementation of quantum information. One major encouragement are the threshold theorems [26] [2] [50] which state that if these components can be engineered with sufficient reliability, then certain known techniques will yield quantum circuits which are extremely reliable in the presence of quantum noise.

However, quantum engineering faces many profoundly difficult challenges, and the timeline for quantum computing is seemingly going to be measured in decades. In the meantime significant effort is underway to more fully understand the power of quantum computers, and to use this information to invent new quantum algorithms.

2.4 A Formalization of Quantum Circuits

This section describes the formalization of quantum circuits employed by this work.

2.4.1 Qubits and Quantum Registers

The fundamental object of quantum information is the quantum bit, or qubit, which represent an abstract quantum mechanical system with two distinct basis states. These states are labeled in correspondence to the 0 and 1 states of a classical bit, and are represented as shown in 2.6.

$$\mathbf{0} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \mathbf{1} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.6}$$

Since a qubit is a quantum mechanical system it can also exist in a superposition of these two basis states, and so the state space $\mathbf{q}$ of a general qubit is as shown in 2.7.

$$\mathbf{q} = \left\{ \alpha_0\mathbf{0} + \alpha_1\mathbf{1} \;\middle|\; \begin{array}{l} \alpha_0, \alpha_1 \in \mathbb{C} \\ |\alpha_0|^2 + |\alpha_1|^2 = 1 \end{array} \right\} = \left\{ \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} \;\middle|\; \begin{array}{l} \alpha_0, \alpha_1 \in \mathbb{C} \\ |\alpha_0|^2 + |\alpha_1|^2 = 1 \end{array} \right\} \tag{2.7}$$

Measurement of a qubit causes the superposition to collapse to a single definite state. Quantum mechanics only provides a probabilistic description of measurement outcomes, as shown in 2.8.

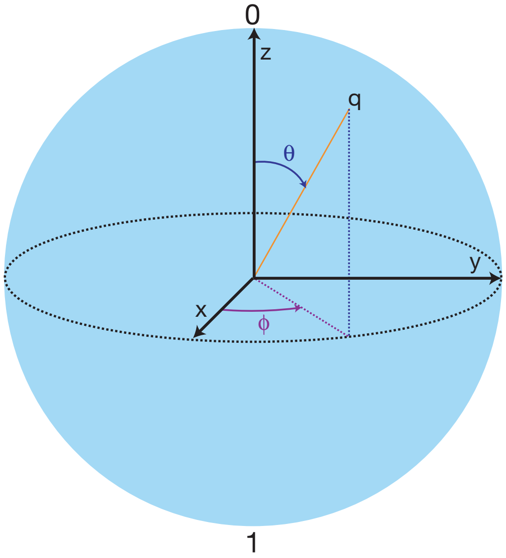
$$\mathbf{M_0} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \qquad \mathbf{M_1} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$
$$\Pr(\mathbf{1} \text{ is observed}) = (\mathbf{M_1}q)^\dagger(\mathbf{M_1}q) = |\alpha_1|^2 \tag{2.8}$$
$$\Pr(\mathbf{0} \text{ is observed}) = (\mathbf{M_0}q)^\dagger(\mathbf{M_0}q) = |\alpha_0|^2$$

Not all qubit states are distinct. Two qubits q_1 and q_2 are global phase equivalent if one can be written as a complex phase multiple of the other, as shown in 2.9. Such qubits cannot be distinguished by any physical procedure, and so they are for all purposes the same state.

$$q_1 \equiv q_2 \iff \exists \theta \in \mathbb{R} \ni q_1 = e^{i\theta} q_2 \tag{2.9}$$

A qubit has another equivalent representation, known as the Bloch Sphere. This geometrical representation is defined by the identity shown in 2.10 which induces the diagram shown in 2.11. The θ degree of freedom determines the probability of observing each of the basis states upon measurement. States located in the upper hemisphere are more likely to obtain a measurement result of **0**, and states located in the lower hemisphere are more likely to obtain **1**. The ϕ degree of freedom is also known as phase. This quantity has no classical counterpart, and can only be observed indirectly, but it does play an important role in the dynamics of qubits.

$$\mathbf{q} = \left\{ \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} \middle| \begin{array}{l} \alpha_0, \alpha_1 \in \mathbb{C} \\ |\alpha_0|^2 + |\alpha_1|^2 = 1 \end{array} \right\} \equiv \left\{ \begin{pmatrix} \cos(\frac{\theta}{2}) \\ e^{i\phi} \sin(\frac{\theta}{2}) \end{pmatrix} \middle| \theta, \phi \in \mathbb{R} \right\} \tag{2.10}$$



(2.11)

Multiple qubits can be joined into a larger system known as a quantum register. The state space $\mathbf{Q}(n)$ of a quantum register containing n qubits is obtained by applying the tensor product to the state spaces of the component qubits, as shown in 2.12.

$$\mathbf{Q}(n) = \bigotimes_{i=0}^{n-1} \mathbf{q}_i \tag{2.12}$$

Each of the n qubits can be observed to be either $\mathbf{0}$ or $\mathbf{1}$, and therefore a quantum register can be observed in any of 2^n many possible configurations. These configurations can be interpreted as the number i in binary, and each configuration corresponds to the quantum register basis state labled $\mathbf{i}$, and is represented by a 2^n component vector with zeros everywhere except a 1 in position i, as shown in 2.13.

$$\begin{array}{ccc}
\textbf{register configuration} & \textbf{state label} & \textbf{state representation} \\
00\ldots00 & \mathbf{0} & \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \\
00\ldots01 & \mathbf{1} & \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \\
\vdots & \vdots & \vdots \\
11\ldots11 & \mathbf{2^n-1} & \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}
\end{array} \tag{2.13}$$

Since a quantum register is a quantum mechanical system it can also exist in a superposition of its 2^n many basis states, and so the state space $\mathbf{Q}(n)$ of a general quantum register can be explicity defined as shown in 2.14.

$$\mathbf{Q}(n) = \bigotimes_{i=0}^{n-1} \mathbf{q}_i = \left\{ \sum_{i=0}^{2^n-1} \alpha_i \, \mathbf{i} \;\middle|\; \begin{array}{l} \alpha_i \in \mathbb{C} \\ \displaystyle\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1 \end{array} \right\} = \left\{ \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{2^n-1} \end{pmatrix} \middle| \begin{array}{l} \alpha_i \in \mathbb{C} \\ \displaystyle\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1 \end{array} \right\} \tag{2.14}$$

Quantum registers fall into the same global phase equivalence classes as qubits. Two quantum registers Q_1 and Q_2 are global phase equivalent if one can be written as a complex phase multiple of the other, as shown in 2.15.

$$Q_1 \equiv Q_2 \iff \exists\, \theta \in \mathbb{R} \ni Q_1 = \mathbb{e}^{\mathbb{i}\theta} Q_2 \tag{2.15}$$

2.4.2 Quantum Gates

Each 2^n by 2^n unitary matrix U defines a quantum transformation of a n qubit quantum register Q, known as a quantum gate. The result of applying U to Q is given by the matrix vector multiplication UQ.

The Bloch Sphere provides an intuitive understanding of the possible transformations of a single qubit, since any possible transformation of points on the sphere is equivalent to a rotation by a particular angle around a particular axis. It is always possible to express any such transformation as the product of up to two rotations around the coordinate system axes, commonly known as $\mathbf{R_x}(\theta)$, $\mathbf{R_y}(\theta)$, and $\mathbf{R_z}(\theta)$.

Quantum gates also fall into the same global equivalence classes as qubits and quantum registers. Two quantum gates U_1 and U_2 are global phase equivalent if one can be written as a complex phase multiple of the other, as shown in 2.16.

$$U_1 \equiv U_2 \iff \exists\, \theta \in \mathbb{R} \ni U_1 = \mathbb{e}^{\mathbb{i}\theta} U_2 \tag{2.16}$$

Perhaps the most commonly used two qubit transformation is the controlled not, or **CNOT**, which performs the quantum analog of classical conditional logic on two qubits. The effect of the **CNOT** depends on the state of the first of the two qubits that it is applied to, known as the control, and only ever acts upon the second qubit, known as the target. If the control is in the **0** state, then both qubits are left unchanged. If the control is in the state **1**, the target qubit is negated by rotating it around the x axis by an angle of π. In the case general case where the control is in a superposition, then the target will be transformed into some superposition of having been rotated and not.

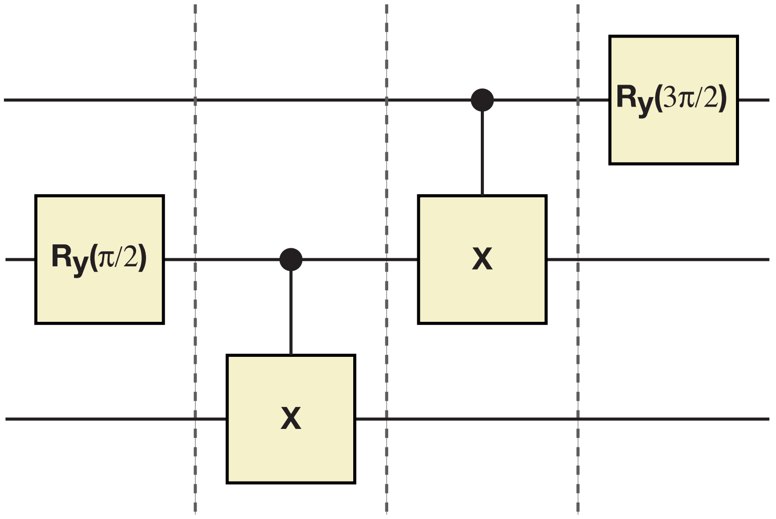
The **TOFFOLI** is a simple generalization of the **CNOT** that operates with two control qubits instead of one. The **TOFFOLI** is significant because Toffoli [15] showed that it was capable of emulating the NAND and FANOUT operations which are sufficient to perform all classical computations. This intuitively establishes that classical computing is a subset of quantum computing.

The definitions of several common quantum gates are to be found in appendix A.

2.4.3 Quantum Circuits

In general, quantum computers are able to apply multiple gates simultaneously, and such a collection of gates in parallel is known as a step. A program for a quantum computer is defined by a sequence of steps, known as a quantum circuit. A quantum circuit C is typically represented in diagramatic form, as shown in 2.17. The transformation $\boldsymbol{\lambda}(C)$ which is effected by C can be represented as a composition of unitary matrices, as shown in 2.18, or the single unitary matrix shown in 2.19. This particular circuit was designed by Brassard [5] to perform the send part of the now well known quantum teleportation experiment.

$$C = \tag{2.17}$$



$$\equiv \boldsymbol{\lambda}(C) = (\mathbb{1}\otimes\mathbf{R_y}(1.57)\otimes\mathbb{1})(\mathbb{1}\otimes\mathbf{CNOT})(\mathbf{CNOT}\otimes\mathbb{1})(\mathbf{R_y}(4.71)\otimes\mathbb{1}\otimes\mathbb{1}) \tag{2.18}$$

$$= \begin{pmatrix} 1 & 0 & -1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & -1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & -1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & -1 & 0 & -1 & 1 & 0 & -1 & 0 \\ -1 & 0 & -1 & 0 & 0 & 1 & 0 & -1 \end{pmatrix} = \mathbf{TELEPORTSEND} \tag{2.19}$$

The effect applying a quantum circuit C to a quantum register Q is obtained by the matrix vector multiplication $\boldsymbol{\lambda}(C)\,Q$. After the circuit is complete, the entire quantum register is observed. The measurement outcome is probabilistic, with the relative probabilities of each configuration as shown in 2.20.

$$\begin{aligned}
&i = b_n b_{n-1} \ldots b_0 \text{ in binary} \\
&\mathbf{M_i} = \mathbf{M}_{b_n} \otimes \mathbf{M}_{b_{n-1}} \otimes \ldots \otimes \mathbf{M}_{b_0} \\
&\Pr(\mathbf{i} \text{ is observed}) = (\mathbf{M_i} Q)^{\dagger} (\mathbf{M_i} Q) = |\alpha_i|^2
\end{aligned} \tag{2.20}$$

Any quantum gate can always be expressed as a circuit composed from certain small sets of gates, known as universal gate sets. In fact, certain small sets of nonparametric gates have been shown to be sufficient to approximate any gate to arbitrary precision [44]. Perhaps the most intuitive universal set is **CNOT** and the single qubit rotations $\mathbf{R_x}(\theta)$, $\mathbf{R_y}(\theta)$, and $\mathbf{R_z}(\theta)$.

2.4.4 Quantum Computers

A particular quantum computer QC is defined by specifying its four components:

- the number of qubits n which QC contains in its quantum register.
- the set of elementary gates E which QC can perform natively.
- a circuit feasibility predicate *feasible?* which determines if QC can implement a given quantum circuit.
- a circuit cost function *cost* which determines the amount of resources required to implement a given quantum circuit on QC.

2.5 The Quantum Circuit Design Problem

Given

A quantum computer with:

- n qubits
- elementary gate set E
- circuit feasibility predicate *feasible?*
- circuit cost function *cost*

A quantum gate G which is to be performed on the quantum computer.

Find

A quantum circuit C such that:

- C is composed only from the gates contained in E
- The transformation $\boldsymbol{\lambda}(C)$ effected by C is G
- *feasible?*(C) is true
- *cost*(C) is minimal

2.5.1 Discussion

The quantum circuit design problem is extremely difficult, for at least the following reasons:

- Since quantum computing contains classical computing, it follows that quantum circuit design is at least as hard as classical circuit design, which is a known hard problem.
- The problem requires the simultaneous solution of two extremely complicated and interdependent subproblems. The first problem is to choose the discrete topology of the circuit, as defined by the assignment of gates to qubits. The second problem is to optimize all of the numerical gate parameters in the chosen circuit topology.
- The assessment of a trial solution requires that a simulation of quantum mechanics be performed, and in general these simulations are believed to necessarily be hard problems for classical computers.
- The local structure of quantum circuits is vastly complicated and poorly understood at best. A small change in the topology or tuning of a circuit can have a large and difficult to predict effect on the feasibility, cost, and transformation that the circuit performs.
- It is unclear what, if anything, can be said in general about minimal circuit costs.

Chapter 3

What is Evolutionary Programming?

> *There is grandeur in this view of life, with its several powers, having been originally breathed into a few forms or into one; and that, whilst this planet has gone cycling on according to the fixed law of gravity, from so simple a beginning endless forms most beautiful and most wonderful have been, and are being, evolved.*
>
> **Charles Darwin [9]**

Despite all of the brilliant successes of algorithms and computing devices there are many interesting problems that remain beyond our reach. Recently, an entirely new approach to problem solving has been studied which seeks to mimic the process of Evolution that has produced the vast and wondrous diversity of life on Earth, nearly all of which is far more complicated than any human artifact. In particular, the human brain is profoundly more powerful than any computer yet built, and is perhaps the most complicated object in the known universe.

3.1 Evolution is a Search Algorithm

Recently we have begun to understand the process of Evolution as a billions of years old search for ever increasingly well adapted forms of life. This understanding is based on the discovery that almost all life is described in the same genetic language which is encoded into configurations of the deoxyribonucleic acid (DNA) molecule. The genetic description of an organism is known as its genotype and is composed of a sequence of genetic building blocks known as genes. Organisms are created via an astonishingly complex developmental process that interprets their genotype as vast collection of sequential and parallel chemical engineering instructions. The collective description of the body and behavior characteristics of an organism is known as its phenotype. The relationships between genotypes and phenotypes are extremely complicated and largely remains a mystery.

Organisms are in a fierce competition for reproductive resources commonly referred to as “the survival of the fittest”. In a process known as natural selection, the amount of reproductive resources an organism receives is proportionate to the relative merits of its phenotype. Therefore the genes associated with successful phenotypes tend to proliferate, and those genes associated with unsuccessful phenotypes are eventually made extinct. The process of natural selection is provided with a continually renewing vast diversity of genotypes for evaluation by the reproductive processes of genetic recombination and mutation.

In the process of genetic recombination, a new child genotype is formed when two parent organisms recombine their genotypes. This recombination process respects their common genotype to phenotype mappings, and so the child inherits a mixture of parental phenotype characteristics. Recombination can be seen as a search through phenotype space which tends to be local to the parents.

The process of genetic mutation results from the action of external chemical, radiological, or physical influences which corrupts the genotype encoded into a DNA molecule. In particular, these mutations tend to occur when a new child genotype is under construction. These mutations may yield small or large structural changes in a genotype depending on the strength of the corrupting influence. Almost all mutations are harmful and many are lethal, but beneficial mutations can occur. Mutation can be seen as a random search through genotype space.

3.2 Evolutionary Programming

The idea of applying evolutionary techniques to problem solving was first explored by Turing in his celebrated 1950 paper [63], in which he suggests that an evolutionary inspired approach to automatic program design as one of three viable directions for artificial intelligence research. Turing was concerned with creating computer programs which are represented using a genetic encoding. Beginning with a seed solution, the vast solution space of possible programs is explored by a sequence of solutions which are obtained by applying small random mutations to the current solution. The effect of each mutation is judged by a human expert, and only those mutations which are deemed to be beneficial are accepted. While Turing never implemented this technique, he proposed that given sufficient time such programs could learn intelligence from their human judges.

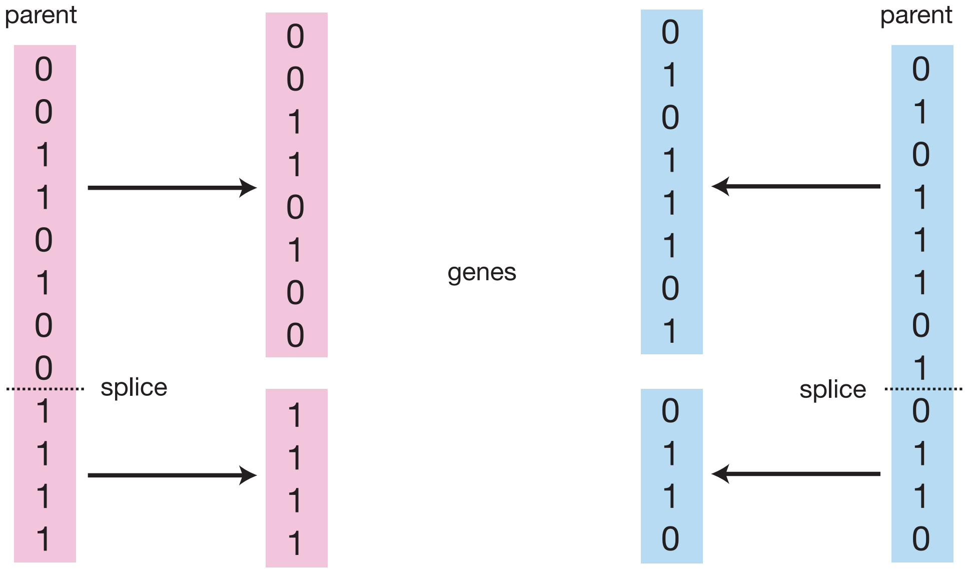
3.2.1 The Genetic Algorithm

The first actual implementation of Evolutionary Programming was the Genetic Algorithm (GA) developed by Holland [21] in 1975. In contrast to Turing's proposal, Holland's work was not directly concerned with designing computer programs, but of finding optimal configurations of fixed length binary vectors which encode solutions to optimization problems. Such a representation is essentially equivalent to DNA molecules, and provides a natural representation for a large class of problems.

Rather than the slow and manual process of merit assessment by a human expert, Holland proposed that the merits of a solution be evaluated by a computational procedure known as the fitness function. Therefore the GA is a fully automatic problem solving system that executes at machine hardware speeds. Even more importantly, Holland proposed that new solutions be obtained not only by a simple random mutation, but in an analogous fashion to genetic recombination. To achieve this the algorithm maintains not just one, but a collection of solutions which are collectively referred to as a population. The population serves as a vastly complicated genetic memory of promising solutions.

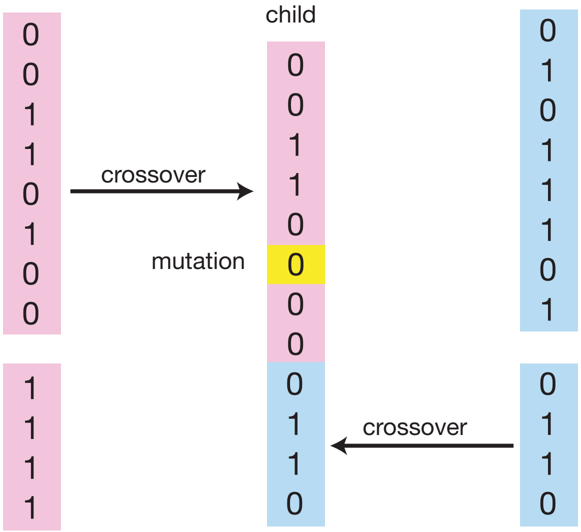
The algorithm begins with an initial population of randomly generated solutions. Each step of the algorithm, known as a generation, reproduces the current population to form a new population for the next generation. Each new population is constructed by directly propagating certain number of solutions from the current population, and by creating a certain number of children by recombination.

Recombination begins by selecting two solutions from the current population to serve as parents. The selection process is performed in accordance with the principle of natural selection in that the probability that a given solution is selected is in some way proportionate to its fitness. Once selected, the parents are spliced into segments, known as genes, as shown in 3.1. The genes serve as the building blocks for the child.



(3.1)

Next, the child is formed by crossing over the genes of the parents into a new genotype, as shown in 3.2.



(3.2)

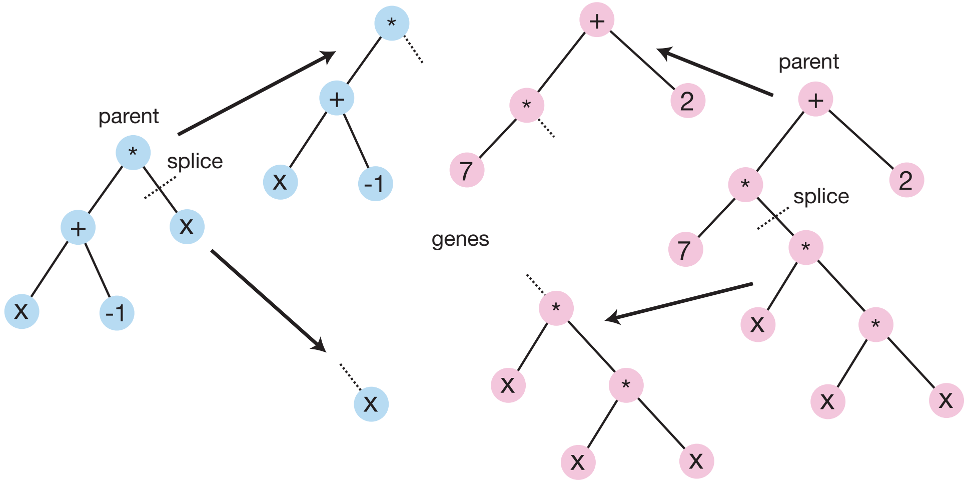
Before the child is added to the new population it is first subjected to a small mutation, typically in the form of a single random gene being flipped with some small probability. Since mutations tend to create children which are very unlike their parents, the process of mutation actually frustrates natural selection. However, mutations serve the critical purpose of continually injecting new genes into the population, and thereby preventing genetic stagnation.

The generational loop continues until a termination condition is reached, at which point the algorithm yields the solution that is best individual in the final population, known as the elite.

3.2.2 Genetic Programming

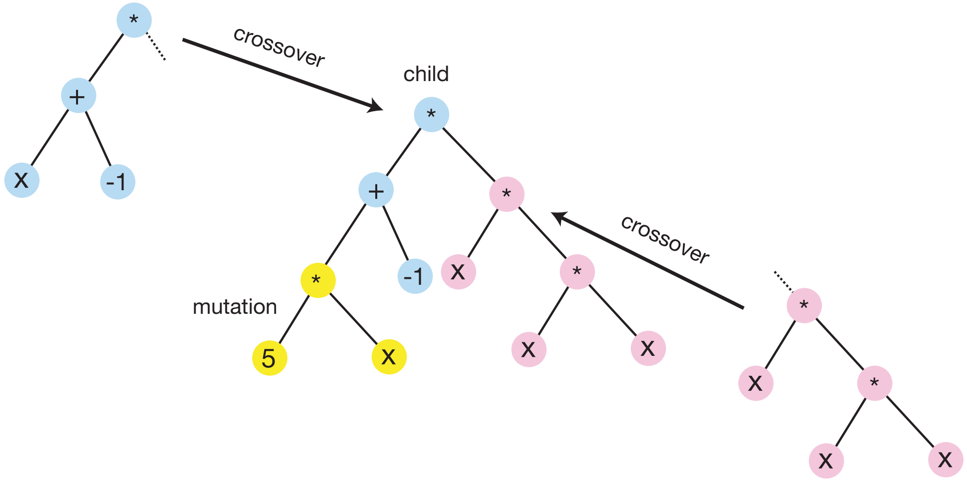
In 1992 Koza [27] proposed Genetic Programming (GP), which makes a single but critical improvement upon the GA by considering the representation of solutions by variable sized tree structures, which are typically used to encode expressions in some formal language. A common representation example is the tree encoding of polynomials shown in 3.3 and 3.4.

Koza extended the procedures for recombination and mutation to the domain of trees. To perform recombination, each parent is first spliced into two subtrees, which serve as genes, by cutting it at a random edge as shown in 3.3.



(3.3)

Next, the child is formed by crossing over the genes to form a new genotype as shown in 3.4. Mutation is performed by randomly replacing some subtree of the child with a random subtree with some low probability.



(3.4)

Koza also developed an extremely simple and efficient parallelization technique for GP, known as the Island Model, which is suitable for implementation on highly scalable and cost effective cluster computers. In this model each node in the cluster is known as an island and maintains its own discrete evolutionary simulation. The sole means of communication between islands occurs when one occasionally broadcasts a small number of randomly chosen individuals from their population which are considered for adoption by their neighboring islands. This low level of communication increases the global genetic diversity since while a local extremum may take over its own island rather quickly, it takes a long time for one solution to take over all other islands, and this gives other solutions a chance to compete.

Since the publication of the first volume, *Genetic Programming* has expanded to 3 additional volumes [28][30][33] which detail advances in Koza's Evolutionary Programming approach, and many reports of successful applications in diverse problem domains.

3.2.3 Memetic Programming

The term Memetic Programming is a very recent invention that broadly refers to any Genetic Programming approach that also incorporate concepts from outside of genetics, such as physics or combinatorial optimization, to form a hybrid approach.

Several recent advances in Evolutionary Programming are in response to an evolutionary theory of culture proposed by Dawkins [10], known as memetics. This theory extends the theory of genetic evolution to the domain of human culture. Dawkins defines the meme to be the unit of imitation in cultural transmission analogous to the gene, and gives examples such as "tunes, ideas, catch-phrases, clothes fashions, ways of making pots or of building arches". Whereas genetic evolution has been searching through configurations of genes that describe organisms that are well adapted to their physical environment, memetic evolution has been searching through configuration of memes that describe ideas that are highly adapted to their cultural environment.

One important distinction between genetic and memetic evolution is that mutations in the genetic domain take the form of small random structural variations, but the mutations in the memetic domain take the form of small nonrandom semantic variations, typically becoming simpler or more expressive. Therefore memetic approaches extends the previous concept of mutation to include not only random structural mutations, but more complicated local search procedures which utilize insight into the problem domain in hopes of performing a beneficial mutation.

Memetic Programming is only a few years old, but has already produced significantly better than previous evolutionary approaches on some problems. Of particular note are the highly competitive results from GASAT [19] algorithm for the boolean satisfiability problem, a problem that had previously been thought poorly suited to evolutionary approaches.

3.3 Why Use Evolutionary Programming?

> *... biological principles, from evolution and genetics to neurobiology and ecology, are informing computer scientist and engineers in designing software and hardware ... and that holds tremendous promise for the future.*
>
> **Rich Lenski** [48]

There are several reasons why an EP approach should be considered as an approach to solving hard problems.

- Simplicity. The basic algorithm is easily understood and can be easily implemented in any programming environment. The application to a new problem domain can be expected to only require a minimal amount of new code to be written.
- The minimal requirement for problem specification. All that is required to apply the algorithm to a specific problem is a representation scheme for solutions that is amenable to recombination and mutation, and a fitness assessment procedure. This minimality of requirements can allow Evolutionary Programming to discover highly novel solutions that would otherwise never be considered.

- The ease of utilization of problem insight. One common technique is to reduce the search space by preventing certain genotype patterns which are believed to be undesirable from ever entering the population. Also, knowledge of the local space of solutions may suggest a memetic style local search mutation. These technique can greatly speed up the evolutionary process.

- Scalability. The vast majority of computational resources are spent on the fitness assessment procedure, and so a reasonable amount of optimization effort can yield excellent run time performance. Also, many fitness assessment procedures can be cost effectively implemented on the commodity hardware that benefits the most from Moore's Law. In addition, the Island Model has proven to be a simple yet extremely effective technique for the distribution of large evolutionary programming problems across a very large cluster of inexpensive computers.

- Evolutionary Programming is an active academic and industrial research area with a large body of successful results. It has been used in such diverse problem domains such as:

    - Bioinformatics, including the problems of drug design, genome sequencing, and protein folding [14].
    - Game theory, including foundational work on the origin of cooperative economic behaviors[3].
    - The arts, including the evolution of aesthetic computer graphics [58], poetry [65], and improvisational jazz [49].
    - Robotics, including the evolution of walking, swimming, and game playing virtual creatures [59], some of which are now being physically constructed [37].
    - Combinatorial optimization, including the highly competitive GASAT algorithm [19] for the SAT problem.
    - Circuit design, which is perhaps the most successful application of Evolutionary Programming [33].

3.4 The Limitations of Evolutionary Programming

Evolutionary Programming is a very intuitive and promising problem solving technique, but there is little formal understanding of how it works and of how to use it most efficiently. Any implementation of an Evolutionary Programming algorithm will require the specification of various parameters, such as population size, mutation rate, and maximum run time, as well as the design of selection, recombination, and mutation procedures. Finding effective choices for these is itself a hard problem with little to no theoretical support. In practice researchers must rely on any available anecdotal reports from related problems, and lots of trial and error.

Chapter 4

Evolutionary Circuit Design

Analog circuit design is known to be a knowledge-intensive, multiphase, iterative task, which usually stretches over a significant period of time and is performed by designers with a large portfolio of skills. It is therefore considered by many to be a form of art rather than a science.

Aaserud and Nielsen [1]

. . . a biologically motivated algorithm (genetic programming) is capable of automatically synthesizing the design of both the topology of complex graphical structures and optimal or near-optimal numerical values for all elements of the structure possessing paramaters.

John R. Koza [29]

4.1 Classical Circuit Design

Koza and others have amassed a sizable body of research [29] [33] [31] [32] which demonstrates that the Genetic Programming approach is capable of automatically synthesizing both the topology and numerical parameters of a variety of forms of computational circuits which meet a set of high level design goals. Their approach entailed:

- A family of simple formal languages to serve as the genotype of the particular kind of circuit, including analog, digital, or mixed electric circuits, neural networks, metabolic pathways, or gene expression networks.
- A genotype to phenotype mapping that translates the high level description of the circuit into native format of the SPICE [51] circuit simulation software with modifications as described in [30].
- A SPICE program which performs fitness assessment by applying a reference signal to the input of the circuit, and measuring the difference between the expected output and the simulated output.

Using this approach, a variety of 20th century circuit design patents were duplicated using relatively little problem insight. Recent work utilizing up to 206 hours of a 1000 node cluster computer resulted in six duplicated patents that were all less than a few years old [34]. The routineness of their approach and the direct benefit of Moore's Law that it enjoys suggest that this approach to circuit design will soon compete with, and possibly surpass, current circuit design strategies, and will yield new patentable inventions.

4.2 Quantum Circuit Design

The application of the Genetic Programming approach to the design of quantum circuits is motivated by the observation that quantum circuits are simply another special case of the family of circuit design problems that have already been studied. All that is required is a formal language for describing the quantum variant of circuit, and a fitness assessment procedure.

The first report on evolutionary quantum circuit design was given by Williams and Gray [66]. They outlined a representation scheme for arbitrary sequential quantum circuits composed from an arbitrary set of elementary quantum gates, including gates with one or more numerical parameters. Their fitness assessment approach utilized the "component wise difference" metric on unitary matrices, shown in 4.1.

$$f(A, B) = \sum_{i,j} |A_{i,j} - B_{i,j}| \tag{4.1}$$

While this fitness assessment does give zero when the two unitary matrices are exactly equal, it does not give zero if two unitary matrices are equivalent up to a global phase factor, and so this fitness assessment does not properly respect global phase equivalences. This means that the algorithm is required to find the exact equivalence class representative specified by the problem, and not any of the uncountably infinitely many other equivalent forms which are physically indistinguishable.

Williams and Gray completed their report with a demonstration of the viability of Genetic Programming by evolving elegant and completely correct quantum circuits to perform the send and receive parts of the quantum teleportation protocol. The search space was reduced by an educated guess that a certain small, non universal, and non parameterized elementary gate would be sufficient.

Rubinstein provided a follow up report [52] which explored a more traditional genetic algorithm approach. This approach represented quantum circuits using variable length bit vectors, and utilized a case based fitness assessment procedure. This report contained successful experimental results which consisted of elegant and completely correct quantum circuits using 2, 3, 4, and 5 qubits. The search space was reduced by an educated guess that it would be sufficient to utilize a non universal elementary gate set consisting of the Hadamard gate, and a generalized form of CNOT which can operate on arbitrary control and target qubits.

The profound successes of the Genetic Programming approach which has routinely delivered high return human competitive solutions to circuit design problems, in conjunction with the successful preliminary reports describing the application of these techniques to quantum circuits, has provided the motivation for the further study of Evolutionary Programming approaches to quantum circuit design.

Chapter 5

wabisabi: A Toolkit for the Evolutionary Design of Quantum Circuits

wabisabi is a toolkit for the implementation of Evolutionary Programming approaches to quantum circuit design problems. The toolkit is designed to be fast, flexible, and easy to use by a wide audience. wabisabi is a japanese term that is not directly translatable, but means essentially "the natural beauty in all things".

The toolkit is written in pure functional scheme [24] targeted for the CHICKEN [67] scheme compiler, and utilizes the high performance BLAS [36] matrix algebra package to perform the core numerical operations at near theoretical peak performance. The software runs on a wide variety of hardware, and includes support for the island model of parallelization.

5.1 Representation

This section contains an specification of the formal language used in this work to describe quantum circuits.

The fundamental object of a quantum circuit is the gate, which is represented as a list beginning with a gate symbol and ending with an optional list of parameters. Several gate expressions and their corresponding unitary representations are seen in 5.1.

$$\begin{aligned} \texttt{(Id)} &\equiv \mathbb{1} \\ \texttt{(Ry 1.57)} &\equiv \mathbf{R_y}(1.57) \\ \texttt{(CNOT)} &\equiv \mathbf{CNOT} \end{aligned} \tag{5.1}$$

A collection of gates operating in parallel is known as step. A step is represented as an association list containing one association per gate operating during that step. The keys in this association are a specification of which qubits in the register that this gate is applied to, known as its domain. The corresponding values are gate expressions that operate on that domain, and each qubit must be involved in exactly one gate per step. Several step expressions and their corresponding unitary representations are shown in 5.2.

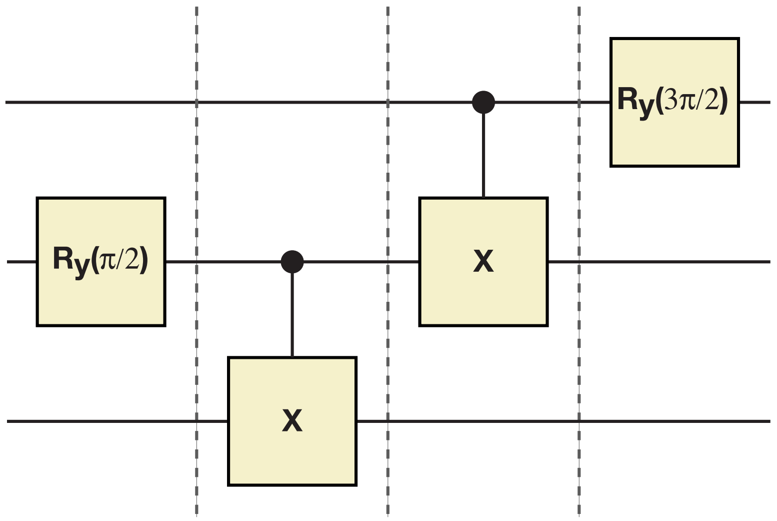
$$\begin{aligned} \texttt{(((1) Id) ((2) Ry 1.57) ((3) Id))} &\equiv \mathbb{1} \otimes \mathbf{R_y}(1.57) \otimes \mathbb{1} \\ \texttt{(((1) Id) ((2 3) CNOT))} &\equiv \mathbb{1} \otimes \mathbf{CNOT} \\ \texttt{(((1 2) CNOT) ((3) Id))} &\equiv \mathbf{CNOT} \otimes \mathbb{1} \\ \texttt{(((1) Ry 4.71) ((2) Id) ((3) Id))} &\equiv \mathbf{R_y}(4.71) \otimes \mathbb{1} \otimes \mathbb{1} \end{aligned} \tag{5.2}$$

A quantum circuit C is simply represented as an ordered list of step expressions, as shown in the example circuit expression in 5.3, which is equivalent to the diagram shown in 5.4. The first step in the circuit is the first element of the circuit expression, and the left most step in the diagram.

The gate that corresponds to the action of C is given by $\boldsymbol{\lambda}(C)$, which can be represented as a composition of unitary matrices as shown in 5.5, which equals the single unitary matrix in 5.6.

$$C = \begin{array}{l}\texttt{((((1) Id) ((2) Ry 1.57) ((3) Id))}\\ \texttt{(((1) Id) ((2 3) CNOT))}\\ \texttt{(((1 2) CNOT) ((3) Id))}\\ \texttt{(((1) Ry 4.71) ((2) Id) ((3) Id)))}\end{array} \tag{5.3}$$

$C \equiv$



(5.4)

$$C \equiv \boldsymbol{\lambda}(C) = (\mathbb{1} \otimes \mathbf{R_y}(1.57) \otimes \mathbb{1})(\mathbb{1} \otimes \mathbf{CNOT})(\mathbf{CNOT} \otimes \mathbb{1})(\mathbf{R_y}(4.71) \otimes \mathbb{1} \otimes \mathbb{1}) \tag{5.5}$$

$$\boldsymbol{\lambda}(C) = \begin{pmatrix} 1 & 0 & -1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & -1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & -1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & -1 & 0 & -1 & 1 & 0 & -1 & 0 \\ -1 & 0 & -1 & 0 & 0 & 1 & 0 & -1 \end{pmatrix} = \mathbf{TELEPORTSEND} \tag{5.6}$$

5.2 Fitness Assessment

Fitness assessment of a quantum circuit C utilizes the following three procedures: *correctness*, *cost*, and *feasible?*.

The *correctness* procedure rates how well C performs the goal gate G. The procedure used in this work calculates the average case fidelity between $\boldsymbol{\lambda}(C)$ and the goal gate G, as shown in 5.7. The values are scaled to range over [0,1] such that higher is better, and a correctness of 1 indicates that $\boldsymbol{\lambda}(C) = G$.

$$\textit{correctness}(C) = \frac{|\mathrm{tr}(G^{\dagger}\boldsymbol{\lambda}(C))|}{2^n} \tag{5.7}$$

An important property of this choice of correctness calculation is that it respects global phase equivalences, as discussed in 2.15. If $\boldsymbol{\lambda}(C)$ and G differ only by a complex phase factor, then C is assigned a correctness of 1, as shown in 5.8.

$$\textit{correctness}(C) = \frac{|\mathrm{tr}(G^{\dagger}\boldsymbol{\lambda}(C))|}{2^n} = \frac{|\mathrm{tr}(G^{\dagger}\mathbb{e}^{\mathbb{i}\theta}G)|}{2^n} = \frac{|\mathbb{e}^{\mathbb{i}\theta}\mathrm{tr}(G^{\dagger}G)|}{2^n} = \frac{|\mathbb{e}^{\mathbb{i}\theta}||\mathrm{tr}(\mathbb{1})|}{2^n} = 1 \tag{5.8}$$

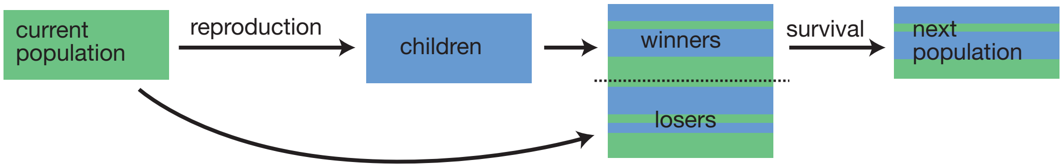
Since the matrices G and $\boldsymbol{\lambda}(C)$ are of size 2^n by 2^n the correctness procedure clearly requires amounts of time and space resources that are exponential in n. However unfortunate, this is to be expected of any procedure that performs a simulation of quantum mechanics. This motivates the use the BLAS libraries to perform highly optimized matrix operations.

The *cost* procedure determines the complexity of implementing C. Perhaps the most basic measures of complexity of a circuit is its length, and a slightly more useful measure is to count the number of non identity gates contained in the circuit. The cost measure used in this work assigns each gate its own particular cost of implementation, and to cost a circuit as the sum of the cost of its gates.

The *feasible?* procedure indicates if a given quantum circuit C is implementable on a given quantum computer. The simple definition of feasibility used in this work is that the circuit has less than some maximum cost, but more than some minimum cost. More complicated definitions can be used to better characterize a particular quantum computer.

5.3 Reproductive Strategy

The reproductive strategy begins with a population of *population size* many randomly generated individuals, each of which contains between 1 and *maximum initial length* steps. The algorithm proceeds by iteratively reproducing the current population, where each iteration represents a generation. Each new population is created by first creating *population size* many children, then merging them with the current population, and finally taking the best *population size* many, as shown in 5.9. The generation loop continues until a termination condition is reached, upon which the best individual of the last population, known as the elite, is returned as the final result.



(5.9)

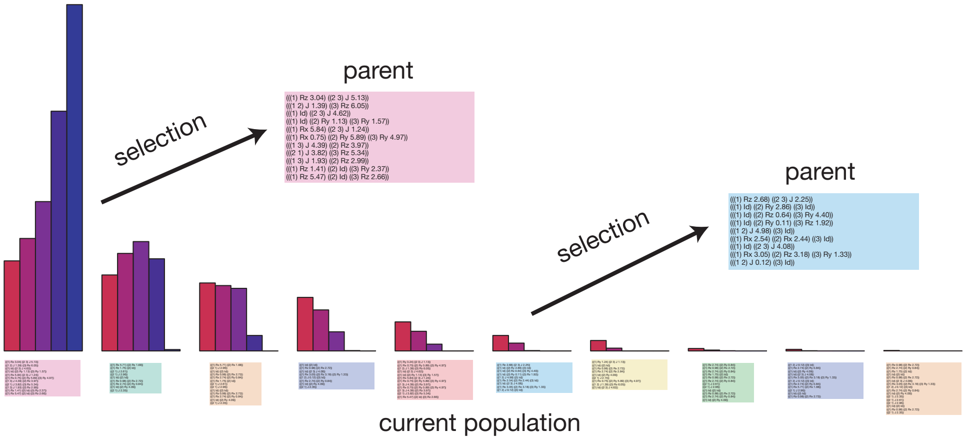
5.3.1 The Temperature Heuristic

The temperature heuristic from the Simulated Annealing [25] algorithm is borrowed in an effort to avoid the local maxima which plague complicated search spaces. Associated with each generation is a temperature that is used to blend two contrasting reproductive strategies, known as exploration and exploitation. The goal of the exploration strategy is to perform a broad search of circuit topologies by performing weakly discriminating selection, random recombination, and only topology affecting mutations. The goal of the exploitation strategy is to optimize the parameters of the topology chosen by the exploration strategy by performing discriminating selection, an order preserving recombination, and only local search mutations which do not affect circuit topology.

The temperature begins as 1, which corresponds to a pure exploration strategy, and after each generation the temperature is slowly cooled by scaling it by a *cooling rate* parameter, but never below the threshold value equal to the correctness of the elite. The threshold serves to guarantee that the algorithm will only attempt to optimize a sufficiently correct solution, and the slow rate of cooling gives the algorithm time to explore many topologies before choosing one for the exploitation strategy.

5.3.2 Selection

The first step in the reproduction process is to select two parents in accordance with natural selection. First the population is sorted by correctness, and then by cost. The parents are chosen from the population according to a temperature dependant approximately normal distribution. The relative probability of selection depends on population rank, as shown in 5.10. At high temperatures, as represented by the redder bars, the selection is liberal, and low ranked individuals have some chance to reproduce. As the temperature cools, as represented by the bluer bars, the selection becomes increasingly more discriminating and only the best individuals are reproduced.


parent
(((1) Rz 3.04) ((2 3) J 5.13))
(((1 2) J 1.39) ((3) Rz 6.05))
(((1) Id) ((2 3) J 4.62))
(((1) Id) ((2) Ry 1.13) ((3) Ry 1.57))
(((1) Rx 5.84) ((2 3) J 1.24))
(((1) Rx 0.75) ((2) Ry 5.89) ((3) Ry 4.97))
(((1 3) J 4.39) ((2) Rz 3.97))
(((2 1) J 3.82) ((3) Rz 5.34))
(((1 3) J 1.93) ((2) Rz 2.99))
(((1) Rz 1.41) ((2) Id) ((3) Ry 2.37))
(((1) Rz 5.47) ((2) Id) ((3) Rz 2.66))
selection
parent
(((1) Rz 2.68) ((2 3) J 2.25))
(((1) Id) ((2) Ry 2.86) ((3) Id))
(((1) Id) ((2) Rz 0.64) ((3) Ry 4.40))
(((1) Id) ((2) Ry 0.11) ((3) Rz 1.92))
(((1 2) J 4.98) ((3) Id))
(((1) Rx 2.54) ((2) Rx 2.44) ((3) Id))
(((1) Id) ((2 3) J 4.08))
(((1) Rx 3.05) ((2) Rz 3.18) ((3) Ry 1.33))
(((1 2) J 0.12) ((3) Id))
selection
current population

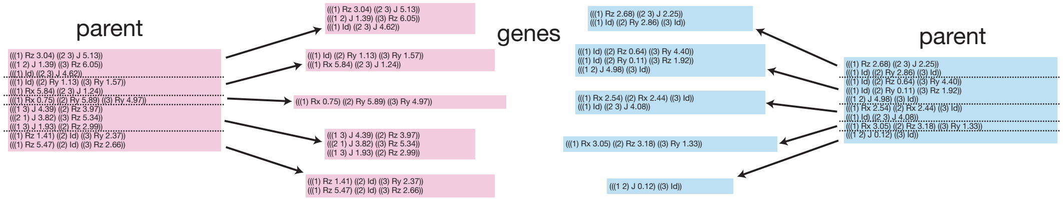
(5.10)

5.3.3 Recombination

Once two parents have been selected they are recombined to form a child. First the parents are spliced into genes, then these genes are recombined, then the recombined genes are subjected to mutation, and finally they are assembled into the completed child.

Splicing

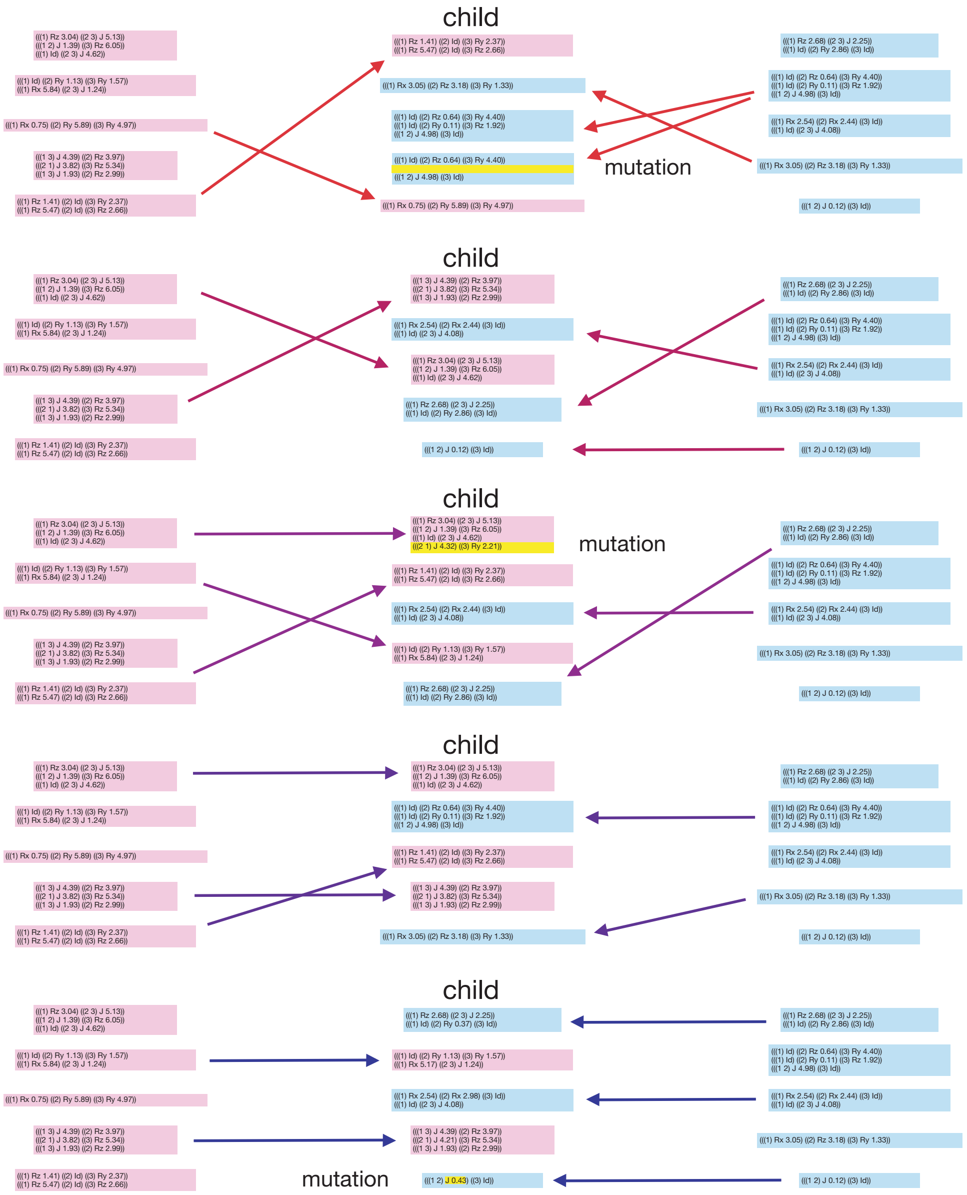
The first step in the recombination process is to splice the parents into the same random number of intervals, as shown in 5.11. These intervals serve as the genes that will be the building blocks used to construct the child.


parent
(((1) Rz 3.04) ((2 3) J 5.13))
(((1 2) J 1.39) ((3) Rz 6.05))
(((1) Id) ((2 3) J 4.62))
(((1) Id) ((2) Ry 1.13) ((3) Ry 1.57))
(((1) Rx 5.84) ((2 3) J 1.24))
(((1) Rx 0.75) ((2) Ry 5.89) ((3) Ry 4.97))
(((1 3) J 4.39) ((2) Rz 3.97))
(((2 1) J 3.82) ((3) Rz 5.34))
(((1 3) J 1.93) ((2) Rz 2.99))
(((1) Rz 1.41) ((2) Id) ((3) Ry 2.37))
(((1) Rz 5.47) ((2) Id) ((3) Rz 2.66))
genes
(((1) Rz 3.04) ((2 3) J 5.13))
(((1 2) J 1.39) ((3) Rz 6.05))
(((1) Id) ((2 3) J 4.62))
(((1) Id) ((2) Ry 1.13) ((3) Ry 1.57))
(((1) Rx 5.84) ((2 3) J 1.24))
(((1) Rx 0.75) ((2) Ry 5.89) ((3) Ry 4.97))
(((1 3) J 4.39) ((2) Rz 3.97))
(((2 1) J 3.82) ((3) Rz 5.34))
(((1 3) J 1.93) ((2) Rz 2.99))
(((1) Rz 1.41) ((2) Id) ((3) Ry 2.37))
(((1) Rz 5.47) ((2) Id) ((3) Rz 2.66))
(((1) Rz 2.68) ((2 3) J 2.25))
(((1) Id) ((2) Ry 2.86) ((3) Id))
(((1) Id) ((2) Rz 0.64) ((3) Ry 4.40))
(((1) Id) ((2) Ry 0.11) ((3) Rz 1.92))
(((1 2) J 4.98) ((3) Id))
(((1) Rx 2.54) ((2) Rx 2.44) ((3) Id))
(((1) Id) ((2 3) J 4.08))
(((1) Rx 3.05) ((2) Rz 3.18) ((3) Ry 1.33))
(((1 2) J 0.12) ((3) Id))
parent
(((1) Rz 2.68) ((2 3) J 2.25))
(((1) Id) ((2) Ry 2.86) ((3) Id))
(((1) Id) ((2) Rz 0.64) ((3) Ry 4.40))
(((1) Id) ((2) Ry 0.11) ((3) Rz 1.92))
(((1 2) J 4.98) ((3) Id))
(((1) Rx 2.54) ((2) Rx 2.44) ((3) Id))
(((1) Id) ((2 3) J 4.08))
(((1) Rx 3.05) ((2) Rz 3.18) ((3) Ry 1.33))
(((1 2) J 0.12) ((3) Id))

(5.11)

Multipoint Crossover

Once the parents have been spliced into a number of genes, the next step is to choose half of them to form the child using a temperature dependent multipoint crossover, as shown in 5.12. The child is constructed in sequence, choosing one gene at a time until the child has the same number of genes as the parents. At high temperatures, as indicated by redder crossover arrows, the genes are chosen randomly from a random parent. At low temperatures, as indicated by the bluer crossover arrows, the genes are chosen from a random parent, but from the position of the parent that corresponds to the current position in the child.


child
(((1) Rz 3.04) ((2 3) J 5.13)) (((1 2) J 1.39) ((3) Rz 6.05)) (((1) Id) ((2 3) J 4.62))
(((1) Id) ((2) Ry 1.13) ((3) Ry 1.57)) (((1) Rx 5.84) ((2 3) J 1.24))
(((1) Rx 0.75) ((2) Ry 5.89) ((3) Ry 4.97))
(((1 3) J 4.39) ((2) Rz 3.97)) (((2 1) J 3.82) ((3) Rz 5.34)) (((1 3) J 1.93) ((2) Rz 2.99))
(((1) Rz 1.41) ((2) Id) ((3) Ry 2.37)) (((1) Rz 5.47) ((2) Id) ((3) Rz 2.66))
(((1) Rz 1.41) ((2) Id) ((3) Ry 2.37)) (((1) Rz 5.47) ((2) Id) ((3) Rz 2.66))
(((1) Rx 3.05) ((2) Rz 3.18) ((3) Ry 1.33))
(((1) Id) ((2) Rz 0.64) ((3) Ry 4.40)) (((1) Id) ((2) Ry 0.11) ((3) Rz 1.92)) (((1 2) J 4.98) ((3) Id))
(((1) Id) ((2) Rz 0.64) ((3) Ry 4.40)) (((1 2) J 4.98) ((3) Id))
(((1) Rx 0.75) ((2) Ry 5.89) ((3) Ry 4.97))
mutation
(((1) Rz 2.68) ((2 3) J 2.25)) (((1) Id) ((2) Ry 2.86) ((3) Id))
(((1) Id) ((2) Rz 0.64) ((3) Ry 4.40)) (((1) Id) ((2) Ry 0.11) ((3) Rz 1.92)) (((1 2) J 4.98) ((3) Id))
(((1) Rx 2.54) ((2) Rx 2.44) ((3) Id)) (((1) Id) ((2 3) J 4.08))
(((1) Rx 3.05) ((2) Rz 3.18) ((3) Ry 1.33))
(((1 2) J 0.12) ((3) Id))
child
(((1) Rz 3.04) ((2 3) J 5.13)) (((1 2) J 1.39) ((3) Rz 6.05)) (((1) Id) ((2 3) J 4.62))
(((1) Id) ((2) Ry 1.13) ((3) Ry 1.57)) (((1) Rx 5.84) ((2 3) J 1.24))
(((1) Rx 0.75) ((2) Ry 5.89) ((3) Ry 4.97))
(((1 3) J 4.39) ((2) Rz 3.97)) (((2 1) J 3.82) ((3) Rz 5.34)) (((1 3) J 1.93) ((2) Rz 2.99))
(((1) Rz 1.41) ((2) Id) ((3) Ry 2.37)) (((1) Rz 5.47) ((2) Id) ((3) Rz 2.66))
(((1 3) J 4.39) ((2) Rz 3.97)) (((2 1) J 3.82) ((3) Rz 5.34)) (((1 3) J 1.93) ((2) Rz 2.99))
(((1) Rx 2.54) ((2) Rx 2.44) ((3) Id)) (((1) Id) ((2 3) J 4.08))
(((1) Rz 3.04) ((2 3) J 5.13)) (((1 2) J 1.39) ((3) Rz 6.05)) (((1) Id) ((2 3) J 4.62))
(((1) Rz 2.68) ((2 3) J 2.25)) (((1) Id) ((2) Ry 2.86) ((3) Id))
(((1 2) J 0.12) ((3) Id))
(((1) Rz 2.68) ((2 3) J 2.25)) (((1) Id) ((2) Ry 2.86) ((3) Id))
(((1) Id) ((2) Rz 0.64) ((3) Ry 4.40)) (((1) Id) ((2) Ry 0.11) ((3) Rz 1.92)) (((1 2) J 4.98) ((3) Id))
(((1) Rx 2.54) ((2) Rx 2.44) ((3) Id)) (((1) Id) ((2 3) J 4.08))
(((1) Rx 3.05) ((2) Rz 3.18) ((3) Ry 1.33))
(((1 2) J 0.12) ((3) Id))
child
(((1) Rz 3.04) ((2 3) J 5.13)) (((1 2) J 1.39) ((3) Rz 6.05)) (((1) Id) ((2 3) J 4.62))
(((1) Id) ((2) Ry 1.13) ((3) Ry 1.57)) (((1) Rx 5.84) ((2 3) J 1.24))
(((1) Rx 0.75) ((2) Ry 5.89) ((3) Ry 4.97))
(((1 3) J 4.39) ((2) Rz 3.97)) (((2 1) J 3.82) ((3) Rz 5.34)) (((1 3) J 1.93) ((2) Rz 2.99))
(((1) Rz 1.41) ((2) Id) ((3) Ry 2.37)) (((1) Rz 5.47) ((2) Id) ((3) Rz 2.66))
(((1) Rz 3.04) ((2 3) J 5.13)) (((1 2) J 1.39) ((3) Rz 6.05)) (((1) Id) ((2 3) J 4.62)) (((2 1) J 4.32) ((3) Ry 2.21))
mutation
(((1) Rz 1.41) ((2) Id) ((3) Ry 2.37)) (((1) Rz 5.47) ((2) Id) ((3) Rz 2.66))
(((1) Rx 2.54) ((2) Rx 2.44) ((3) Id)) (((1) Id) ((2 3) J 4.08))
(((1) Id) ((2) Ry 1.13) ((3) Ry 1.57)) (((1) Rx 5.84) ((2 3) J 1.24))
(((1) Rz 2.68) ((2 3) J 2.25)) (((1) Id) ((2) Ry 2.86) ((3) Id))
(((1) Rz 2.68) ((2 3) J 2.25)) (((1) Id) ((2) Ry 2.86) ((3) Id))
(((1) Id) ((2) Rz 0.64) ((3) Ry 4.40)) (((1) Id) ((2) Ry 0.11) ((3) Rz 1.92)) (((1 2) J 4.98) ((3) Id))
(((1) Rx 2.54) ((2) Rx 2.44) ((3) Id)) (((1) Id) ((2 3) J 4.08))
(((1) Rx 3.05) ((2) Rz 3.18) ((3) Ry 1.33))
(((1 2) J 0.12) ((3) Id))
child
(((1) Rz 3.04) ((2 3) J 5.13)) (((1 2) J 1.39) ((3) Rz 6.05)) (((1) Id) ((2 3) J 4.62))
(((1) Id) ((2) Ry 1.13) ((3) Ry 1.57)) (((1) Rx 5.84) ((2 3) J 1.24))
(((1) Rx 0.75) ((2) Ry 5.89) ((3) Ry 4.97))
(((1 3) J 4.39) ((2) Rz 3.97)) (((2 1) J 3.82) ((3) Rz 5.34)) (((1 3) J 1.93) ((2) Rz 2.99))
(((1) Rz 1.41) ((2) Id) ((3) Ry 2.37)) (((1) Rz 5.47) ((2) Id) ((3) Rz 2.66))
(((1) Rz 3.04) ((2 3) J 5.13)) (((1 2) J 1.39) ((3) Rz 6.05)) (((1) Id) ((2 3) J 4.62))
(((1) Id) ((2) Rz 0.64) ((3) Ry 4.40)) (((1) Id) ((2) Ry 0.11) ((3) Rz 1.92)) (((1 2) J 4.98) ((3) Id))
(((1) Rz 1.41) ((2) Id) ((3) Ry 2.37)) (((1) Rz 5.47) ((2) Id) ((3) Rz 2.66))
(((1 3) J 4.39) ((2) Rz 3.97)) (((2 1) J 3.82) ((3) Rz 5.34)) (((1 3) J 1.93) ((2) Rz 2.99))
(((1) Rx 3.05) ((2) Rz 3.18) ((3) Ry 1.33))
(((1) Rz 2.68) ((2 3) J 2.25)) (((1) Id) ((2) Ry 2.86) ((3) Id))
(((1) Id) ((2) Rz 0.64) ((3) Ry 4.40)) (((1) Id) ((2) Ry 0.11) ((3) Rz 1.92)) (((1 2) J 4.98) ((3) Id))
(((1) Rx 2.54) ((2) Rx 2.44) ((3) Id)) (((1) Id) ((2 3) J 4.08))
(((1) Rx 3.05) ((2) Rz 3.18) ((3) Ry 1.33))
(((1 2) J 0.12) ((3) Id))
child
(((1) Rz 3.04) ((2 3) J 5.13)) (((1 2) J 1.39) ((3) Rz 6.05)) (((1) Id) ((2 3) J 4.62))
(((1) Id) ((2) Ry 1.13) ((3) Ry 1.57)) (((1) Rx 5.84) ((2 3) J 1.24))
(((1) Rx 0.75) ((2) Ry 5.89) ((3) Ry 4.97))
(((1 3) J 4.39) ((2) Rz 3.97)) (((2 1) J 3.82) ((3) Rz 5.34)) (((1 3) J 1.93) ((2) Rz 2.99))
(((1) Rz 1.41) ((2) Id) ((3) Ry 2.37)) (((1) Rz 5.47) ((2) Id) ((3) Rz 2.66))
(((1) Rz 2.68) ((2 3) J 2.25)) (((1) Id) ((2) Ry 0.37) ((3) Id))
(((1) Id) ((2) Ry 1.13) ((3) Ry 1.57)) (((1) Rx 5.17) ((2 3) J 1.24))
(((1) Rx 2.54) ((2) Rx 2.98) ((3) Id)) (((1) Id) ((2 3) J 4.08))
(((1 3) J 4.39) ((2) Rz 3.97)) (((2 1) J 4.21) ((3) Rz 5.34)) (((1 3) J 1.93) ((2) Rz 2.99))
mutation
(((1 2) J 0.43) ((3) Id))
(((1) Rz 2.68) ((2 3) J 2.25)) (((1) Id) ((2) Ry 2.86) ((3) Id))
(((1) Id) ((2) Rz 0.64) ((3) Ry 4.40)) (((1) Id) ((2) Ry 0.11) ((3) Rz 1.92)) (((1 2) J 4.98) ((3) Id))
(((1) Rx 2.54) ((2) Rx 2.44) ((3) Id)) (((1) Id) ((2 3) J 4.08))
(((1) Rx 3.05) ((2) Rz 3.18) ((3) Ry 1.33))
(((1 2) J 0.12) ((3) Id))

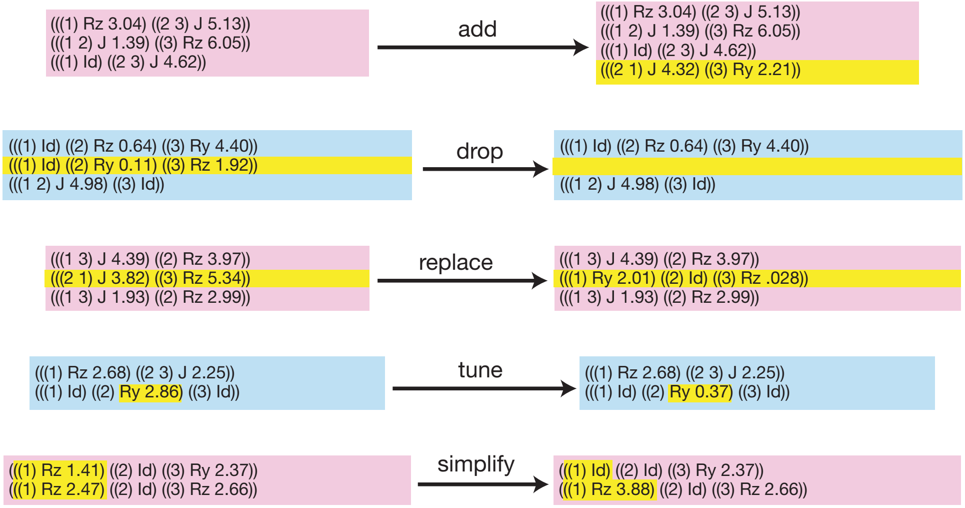
(5.12)

Mutations

Each gene inherited by a child is potentially subjected to one of several mutations with a frequency determined by the mutation rate parameter, as shown in 5.13.

At high temperatures only the *add*, *drop*, and *replace* mutations tend to be applied. The *add* mutation inserts a new random step into the gene, the *drop* mutation removes a random step from the gene, and the *replace* mutation first removes a random step from the gene and then inserts a new random step in its place. These mutations perform small but topologically significant changes to the gene which are used by the exploration strategy to find improved circuit topologies.

At low temperatures, only the *tune* and *simplify* mutations are applied. The *tune* mutation selects a random parameter in the gene and perturbs it by a small amount, and the *simplify* mutation performs algebraic simplification on the gene in an effort to reduce its cost. These mutations are utilized by the exploitation strategy in the attempt to optimize the topology of a circuit.


(((1) Rz 3.04) ((2 3) J 5.13))
(((1 2) J 1.39) ((3) Rz 6.05))
(((1) Id) ((2 3) J 4.62))
add
(((1) Rz 3.04) ((2 3) J 5.13))
(((1 2) J 1.39) ((3) Rz 6.05))
(((1) Id) ((2 3) J 4.62))
(((2 1) J 4.32) ((3) Ry 2.21))
(((1) Id) ((2) Rz 0.64) ((3) Ry 4.40))
(((1) Id) ((2) Ry 0.11) ((3) Rz 1.92))
(((1 2) J 4.98) ((3) Id))
drop
(((1) Id) ((2) Rz 0.64) ((3) Ry 4.40))
(((1 2) J 4.98) ((3) Id))
(((1 3) J 4.39) ((2) Rz 3.97))
(((2 1) J 3.82) ((3) Rz 5.34))
(((1 3) J 1.93) ((2) Rz 2.99))
replace
(((1 3) J 4.39) ((2) Rz 3.97))
(((1) Ry 2.01) ((2) Id) ((3) Rz .028))
(((1 3) J 1.93) ((2) Rz 2.99))
(((1) Rz 2.68) ((2 3) J 2.25))
(((1) Id) ((2) Ry 2.86) ((3) Id))
tune
(((1) Rz 2.68) ((2 3) J 2.25))
(((1) Id) ((2) Ry 0.37) ((3) Id))
(((1) Rz 1.41) ((2) Id) ((3) Ry 2.37))
(((1) Rz 2.47) ((2) Id) ((3) Rz 2.66))
simplify
(((1) Id) ((2) Id) ((3) Ry 2.37))
(((1) Rz 3.88) ((2) Id) ((3) Rz 2.66))

(5.13)

Assembly

The final stage of the construction of the child is to assemble it from its list of genes into a new individual. Before a child is added to the child population it is tested by the *feasible?* predicate. If the child is feasible then it has its correctness and cost assessed and is added to the child population, otherwise the child is discarded and the reproductive process tries again.

5.4 Termination Conditions

The generation loop continues until one of three termination conditions is met.

- The success condition occurs if the elite of the population is sufficiently correct, as specified by the *goal correctness* parameter.
- A failure condition occurs if a sufficient amount of time has elapsed since the beginning of the experiment, as specified by the *maximum time* parameter. This condition guarantees an upper bound the amount of time resources that any one experiment may consume.
- A failure condition also occurs if the temperature remains at the threshold level for a sufficiently large number of generations, as specified by the *maximum dwell* parameter. This termination condition is an attempt to recognize when the algorithm has become stuck in a local maximum and is not worthy of further computational resources.

5.5 Algorithmic Parameters

The algorithm depends on the specification of the following parameters.

- G is the goal quantum gate.
- *elementary gate set* determines which elementary gates will be used in the construction of quantum circuits. Each of these gates must have an associated cost, to be used by the *feasible?* predicate.
- *population size* determines the number of individuals contained in the population. Very little is known in general about optimal population sizes. Larger values surely increase genetic diversity which can avoid premature convergence on local extremum, but also slows down the algorithm. A minimal useful value appears to be 100, and some experiments use millions.
- *maximum individual length* determines the largest quantum circuit allowed in the initial population. Longer values increase genetic diversity but can slow down the algorithm.
- *mutation rate* determines the frequency of mutation during the reproductive process. Commonly used values are in the range between 0.001 and 0.2, and sufficiently higher values will reduce the algorithm to a blind search.
- *cooling rate* is used in conjunction with the fitness of the elite to determine the temperature for the next generation. Commonly used values are in the range between .9 and .999.
- *minimum cost* is used by the *feasible?* predicate to prevent solutions which are too simple from entering the population.
- *maximum cost* is used by the *feasible?* predicate to prevent solutions which are too complicated from entering the population.
- *goal correctness* determines the minimum correctness for a success condition.

- *maximum time* determines the maximum amount of time that may elapse during an experiment before a failure condition occurs. This can be used to ensure that a minimum number of experiments are performed in a fixed time.

- *maximum dwell* determines the maximum number of successive generations that may occur such that the temperature remains at the threshold level before a failure condition occurs. Values in the range of 100 to 1000 have been useful.

Chapter 6

Experiments

This section reports on several experiments that were performed in order to test the viability of the evolutionary approach to practical quantum circuit design problems. All experiments utilized a single 2.6 Ghz Pentium4 system, and required no more than 10MB of RAM.

Each experiment consists of several trials, and the trial results are summarized by a scatter plot diagram of cost against trial duration. The red dots represent trials which were terminated by a failure condition, and the green dots represent those which were terminated by a success condition.

For all experiments, *goal correctness* was set to 0.999 and the *minimum cost* was 10. The *maximum cost* and the *maximum time* were set sufficiently high as to not be a factor.

Each experiment utilized one of three elementary gate sets. The **liquid state NMR set** shown in 6.1 represents a generic liquid state nuclear magnetic resonance quantum computer [64], which features discrete X and Y rotations, an inexpensive parameterized Z rotation, and an expensive J gate.

Gate	**Cost**
1	0
$\mathbf{X_1}$	1
$\mathbf{Y_1}$	1
$\mathbf{R_z}(\theta)$	.1
$\mathbf{J}(\theta)$	10

(6.1)

The **standard universal set** shown in 6.2 contains the parameterized rotations and an expensive CNOT gate.

Gate	**Cost**
1	0
$\mathbf{R_x}(\theta)$	1
$\mathbf{R_y}(\theta)$	1
$\mathbf{R_z}(\theta)$	.1
CNOT	10

(6.2)

The **parameterized universal set** shown in 6.3 contains the parameterized rotations and an expensive J gate.

Gate	**Cost**
1	0
$\mathbf{R_x}(\theta)$	1
$\mathbf{R_y}(\theta)$	1
$\mathbf{R_z}(\theta)$	1
$\mathbf{J}(\theta)$	10

(6.3)

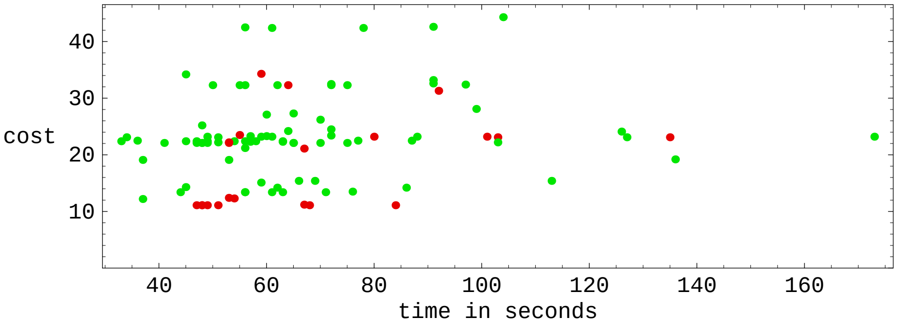
The definitions of all quantum gates are to be found in Appendix A.

6.1 CNOT

This experiment was performed with the experimental parameters as described in 6.4.

$$\begin{array}{ll} \text{G} = & \textbf{CNOT} \\ \text{Elementary Gate Set} = & \text{liquid state NMR} \\ \text{Population Size} = & 500 \\ \text{Maximum Initial Length} = & 10 \\ \text{Maximum Dwell} = & 100 \\ \text{Mutation Rate} = & .2 \\ \text{Cooling Rate} = & .95 \end{array} \tag{6.4}$$

A total of 100 experiments were performed and the results are summarized in 6.5. The success rate was .79 and the average experiment duration was 67 seconds.

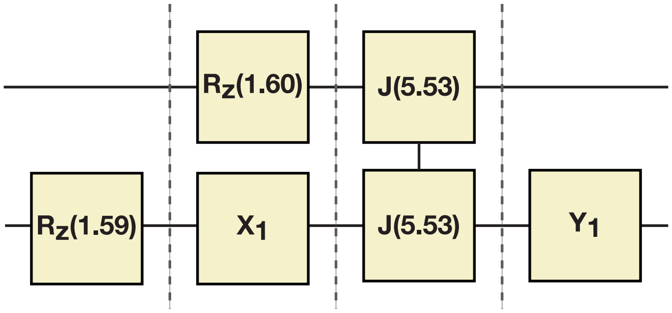


(6.5)

The best circuit over all experiments has a cost of 12.2 and is shown in 6.6 and 6.7.

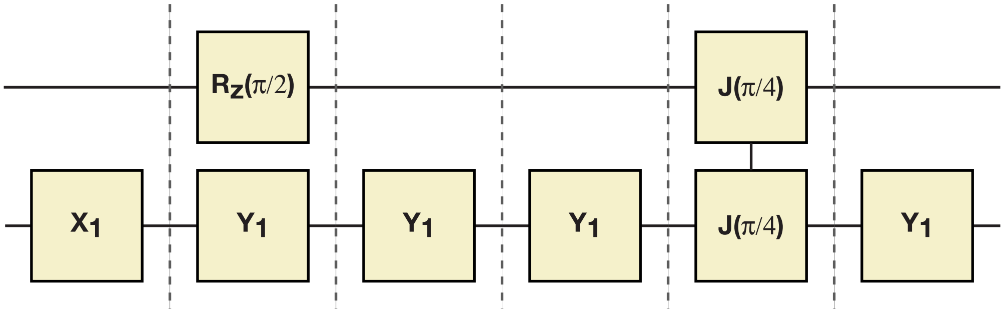
```
((((1) Id) ((2) Rz 1.59308814194286))
(((1) Rz 1.59746615368844) ((2) X1))
(((2 1) J 5.53125259570381))
(((1) Id) ((2) Y1)))
```

(6.6)



(6.7)

One commonly utilized circuit [35], which has a cost of 15.1, is shown in 6.8.



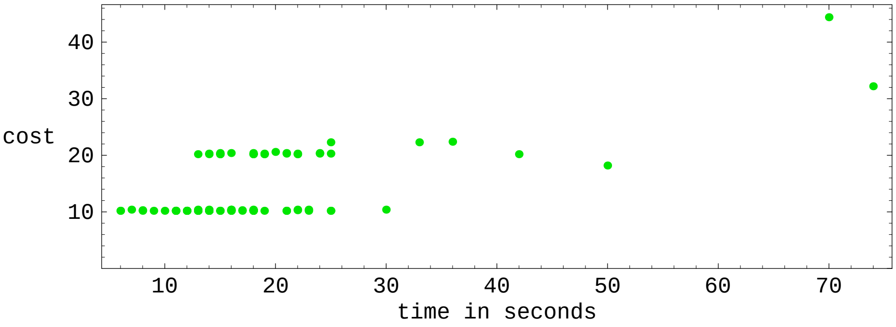
(6.8)

6.2 CPHASE

This experiment was performed with the experimental parameters as described in 6.9.

G =	**CPHASE**	
Elementary Gate Set =	liquid state NMR	
Population Size =	500	
Maximum Initial Length =	10	(6.9)
Maximum Dwell =	100	
Mutation Rate =	.2	
Cooling Rate =	.95	

A total of 100 experiments were performed and the results are summarized in 6.10. The success rate was 1.0 and the average experiment duration was 18 seconds.

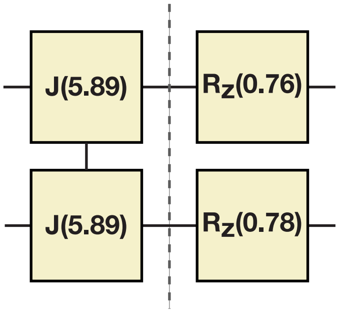


(6.10)

The best circuit over all experiments has a cost of 10.2 and is shown in 6.11 and 6.12.

```
((((1 2) J 5.89156133505902))
(((1) Rz 0.755255117203258) ((2) Rz 0.776411795935828)))
```

(6.11)



(6.12)

There is no commonly utilized circuit for comparison, but since the CSIGN causes a nontrivial interaction between the qubits it operates on, any CSIGN circuit must contain at least one J gate, and so this result is certainly near optimal.

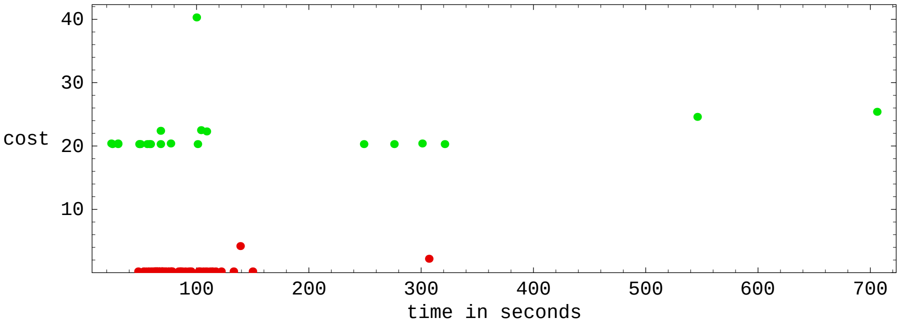
6.3 CPHASE 2

This experiment was performed with the experimental parameters as described in 6.13.

G =	**CPHASE**
Elementary Gate Set =	standard universal
Population Size =	500
Maximum Initial Length =	20
Maximum Dwell =	200
Mutation Rate =	.2
Cooling Rate =	.99

(6.13)

A total of 122 experiments were performed and the results are summarized in 6.14. The success rate was .19 and the average experiment duration was 93.7 seconds.

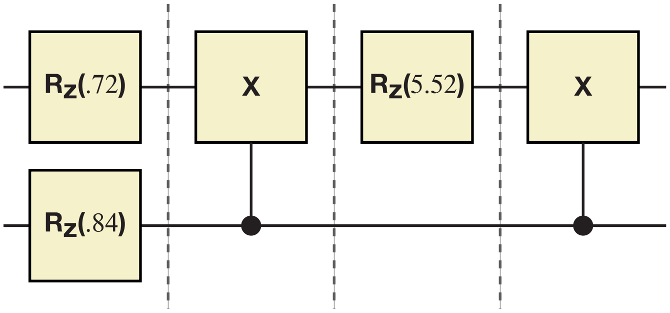


(6.14)

The best circuit over all experiments has a cost of 20.3 and is shown in 6.15 and 6.16.

```
(((1) Rz 0.718624942221927) ((2) Rz 0.839817285381341))
(((2 1) CNOT))
(((1) Rz 5.51723684487596) ((2) Id))
(((2 1) CNOT))
```

(6.15)



(6.16)

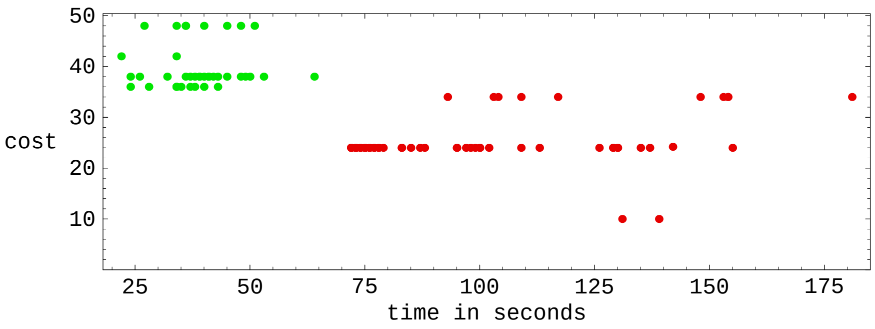
There is no commonly utilized circuit for comparison. There may exist a circuit containing only a single CNOT.

6.4 SWAP

This experiment was performed with the experimental parameters as described in 6.17.

$$\begin{array}{ll}
\text{G} = & \textbf{SWAP} \\
\text{Elementary Gate Set} = & \text{liquid state NMR} \\
\text{Population Size} = & 500 \\
\text{Maximum Initial Length} = & 10 \\
\text{Maximum Dwell} = & 100 \\
\text{Mutation Rate} = & .2 \\
\text{Cooling Rate} = & .95
\end{array} \tag{6.17}$$

A total of 100 experiments were performed and the results are summarized in 6.18. The success rate was 0.81 and the average experiment duration was 76 seconds.

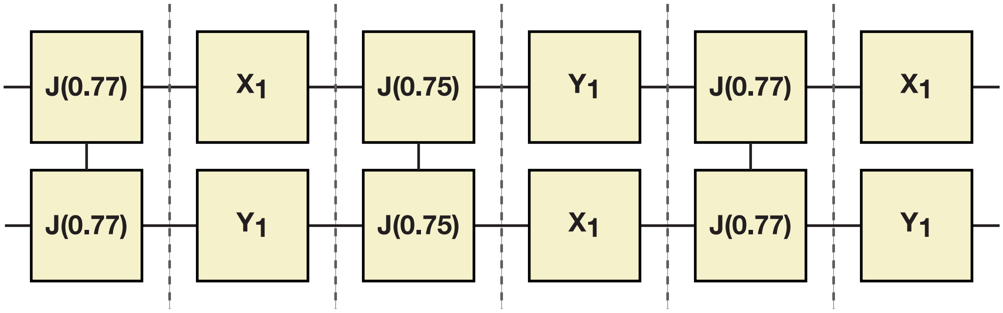


(6.18)

The best circuit over all experiments has a cost of 36.0 and is shown in 6.19 and 6.20.

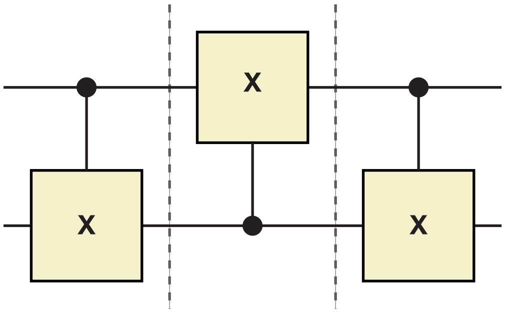
```
((((1 2) J 0.769001845694298))
(((1) X1) ((2) Y1))
(((1 2) J 0.746729562208523))
(((1) Y1) ((2) X1))
(((1 2) J 0.773203898342641))
(((1) X1) ((2) Y1)))
```

(6.19)



(6.20)

The commonly utilized circuit [45] is shown in 6.21. Using the CNOT circuit obtained in 6.6 this circuit has a cost of 36.6.



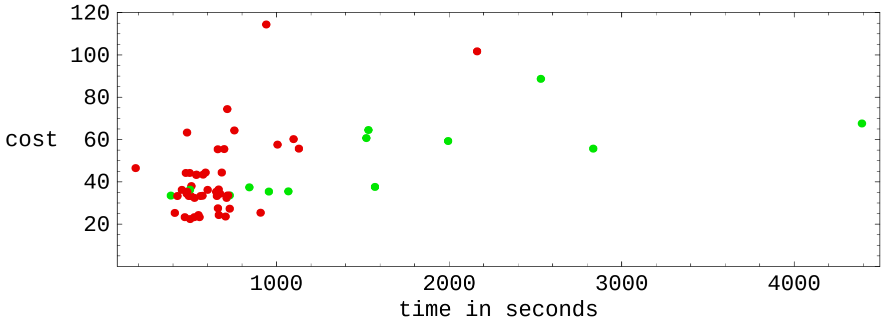
(6.21)

6.5 QFT2

This experiment was performed with the experimental parameters as described in 6.22.

G =	**QFT2**	
Elementary Gate Set =	standard universal	
Population Size =	500	
Maximum Initial Length =	20	(6.22)
Maximum Dwell =	100	
Mutation Rate =	.2	
Cooling Rate =	.99	

A total of 62 experiments were performed and the results are summarized in 6.23. The success rate was 0.21 and the average experiment duration was 848 seconds.

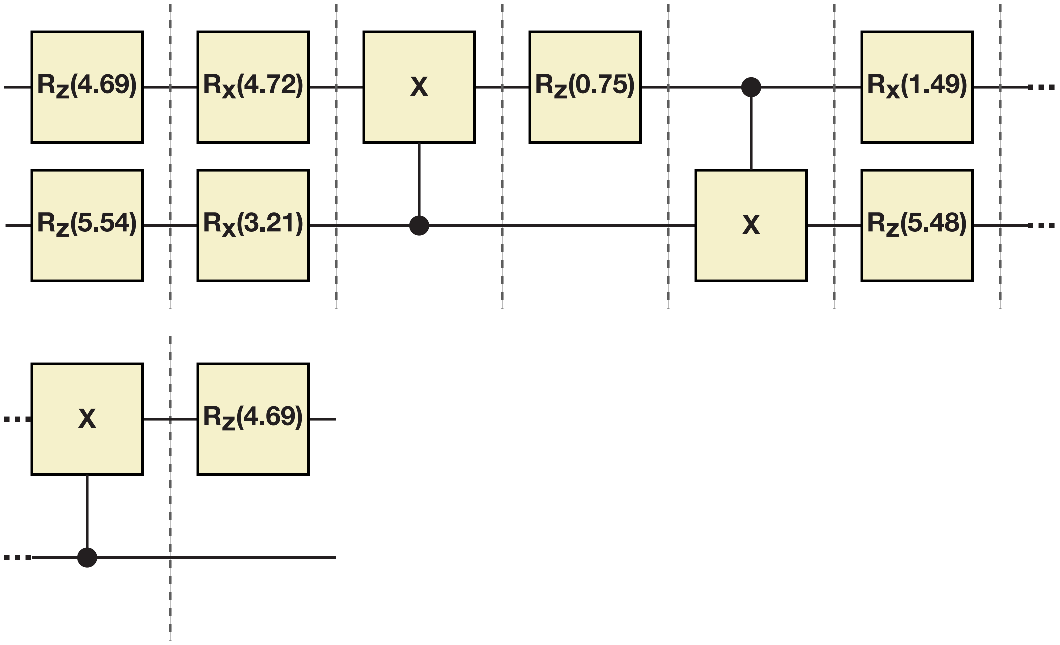


(6.23)

The best circuit over all experiments has a cost of 33.5 and is shown in 6.24 and 6.25.

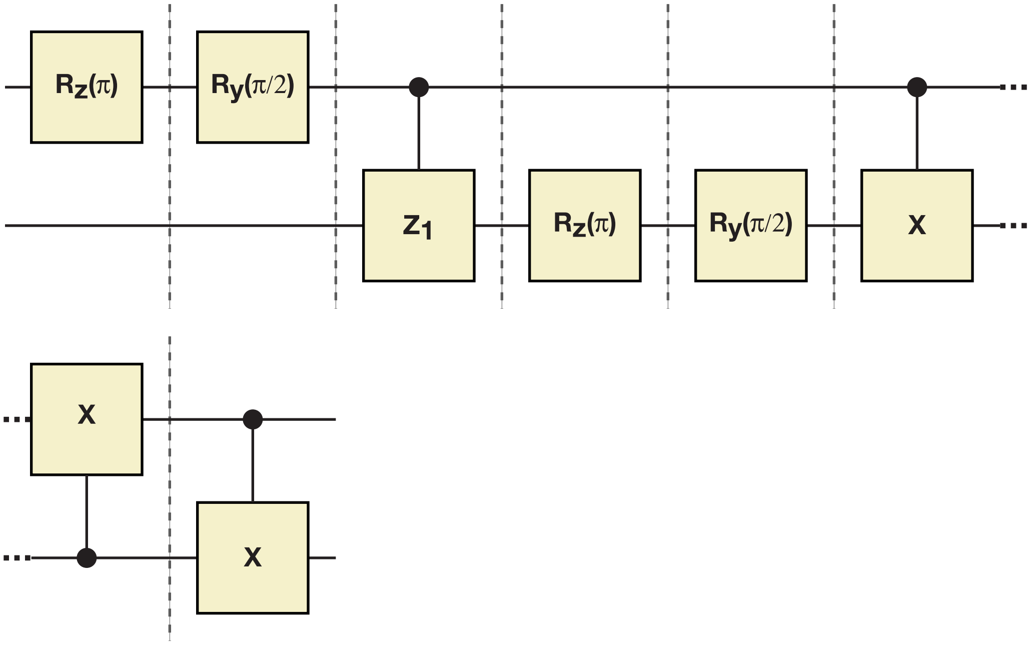
```
((((1) Rz 4.68604458152085) ((2) Rz 5.53675597122988))
(((1) Rx 4.71780632828402) ((2) Rx 3.21191271479259))
(((2 1) CNOT))
(((1) Rz 0.753993811897859) ((2) Id))
(((1 2) CNOT))
(((1) Rx 1.48863515925529) ((2) Rz 5.47527777340135))
(((2 1) CNOT))
(((1) Rz 4.6855792798612) ((2) Id)))
```

(6.24)



(6.25)

The commonly utilized circuit [46] is shown in 6.26. Using the CPHASE circuit obtained in 6.15 this circuit has a cost of 55.2.



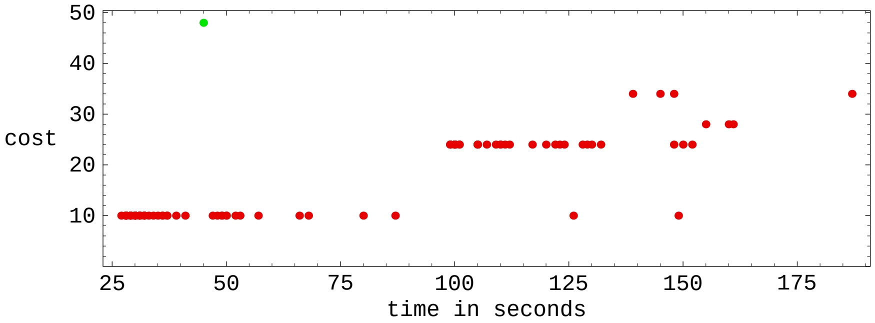
(6.26)

6.6 SQRTSWAP

This experiment was performed with the experimental parameters as described in 6.27.

$$\begin{array}{ll} \text{G} = & \textbf{SQRTSWAP} \\ \text{Elementary Gate Set} = & \text{liquid state NMR} \\ \text{Population Size} = & 500 \\ \text{Maximum Initial Length} = & 10 \\ \text{Maximum Dwell} = & 100 \\ \text{Mutation Rate} = & .2 \\ \text{Cooling Rate} = & .95 \end{array} \tag{6.27}$$

A total of 100 experiments were performed and the results are summarized in 6.28. The success rate was 0.01 and the average experiment duration was 76 seconds.

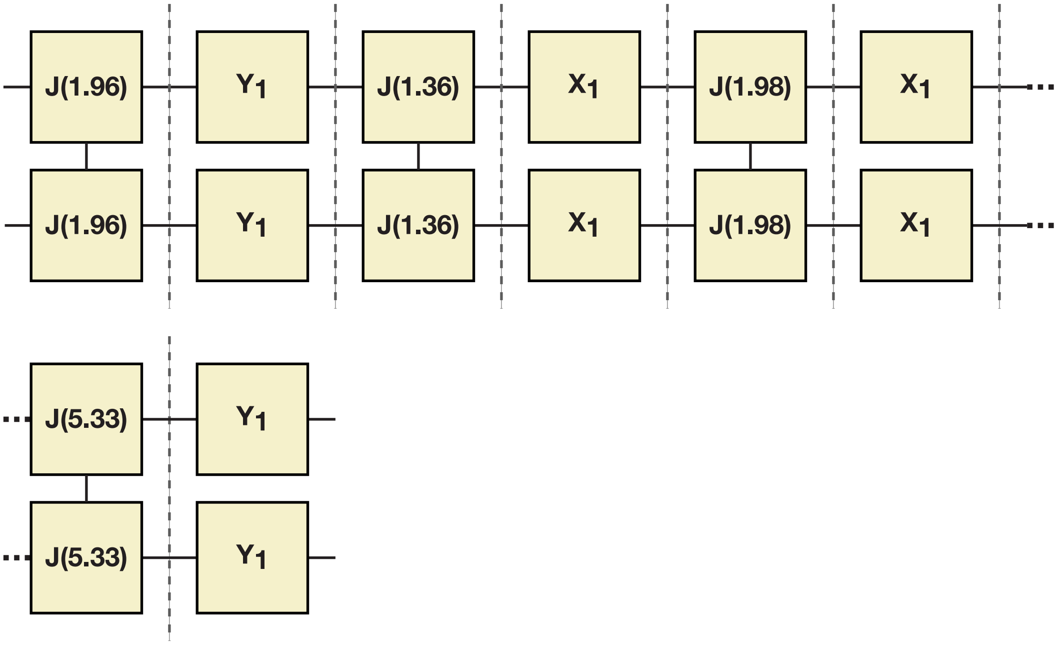


(6.28)

The best circuit over all experiments has a cost of 48.0 and is shown in 6.29 and 6.30.

```
(((1 2) J 1.96145491877727))
(((1) Y1) ((2) Y1))
(((1 2) J 1.3585650864218))
(((1) X1) ((2) X1))
(((2 1) J 1.98073896253607))
(((1) X1) ((2) X1))
(((1 2) J 5.32633592052147))
(((1) Y1) ((2) Y1))
```

(6.29)



(6.30)

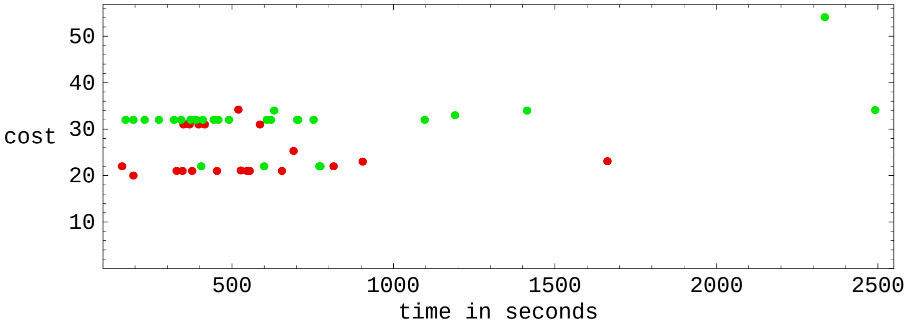
There is no common circuit for comparison. However, the application of two SQRTSWAP gates acts exactly as the SWAP gate at a cost of 96, but the SWAP circuit found in 6.19 has a cost of 36, and so it would seem that this result is far from optimal.

6.7 TELEPORTSEND

This experiment was performed with the experimental parameters as described in 6.31.

G =	**TELEPORTSEND**	
Elementary Gate Set =	standard universal	
Population Size =	500	
Maximum Initial Length =	20	(6.31)
Maximum Dwell =	100	
Mutation Rate =	.2	
Cooling Rate =	.99	

A total of 50 experiments were performed and the results are summarized in 6.32. The success rate was 0.58 and the average experiment duration was 623 seconds.

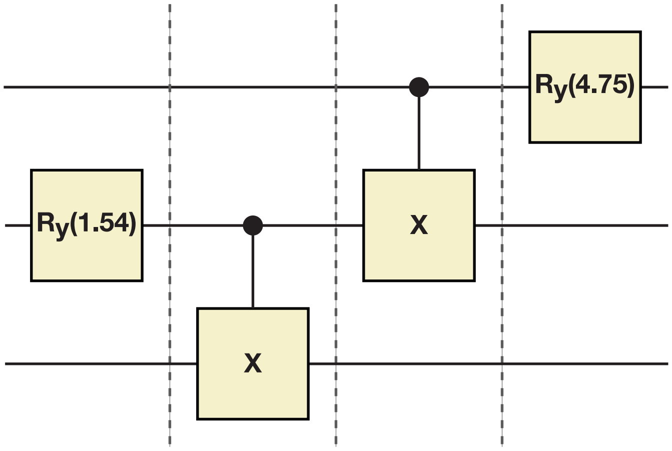


(6.32)

The best circuit over all experiments has a cost of 22.0 and is shown in 6.33 and 6.34.

```
((((1) Id) ((2) Ry 1.5360537360157) ((3) Id))
(((1) Id) ((2 3) CNOT))
(((1 2) CNOT) ((3) Id))
(((1) Ry 4.74616188483069) ((2) Id) ((3) Id)))
```

(6.33)



(6.34)

This is exactly the same circuit as the original invention by Brassard, as shown in 2.18. This circuit was also obtained by Williams and Gray [66].

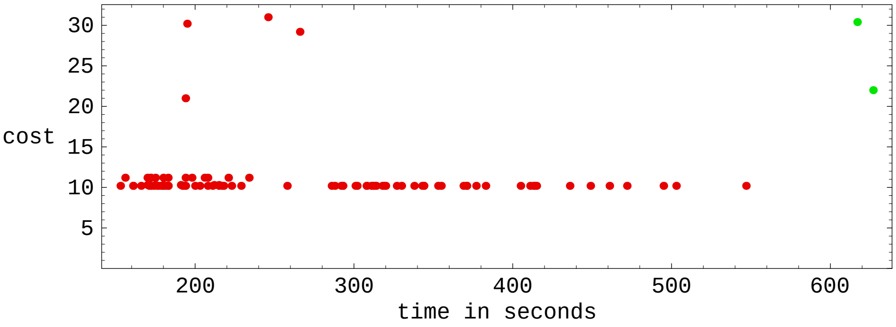
6.8 TELEPORTRECEIVE

This experiment was performed with the experimental parameters as described in 6.35.

G =	**TELEPORTRECEIVE**
Elementary Gate Set =	standard universal
Population Size =	500
Maximum Initial Length =	20
Maximum Dwell =	100
Mutation Rate =	.2
Cooling Rate =	.99

(6.35)

A total of 100 experiments were performed and the results are summarized in 6.36. The success rate was 0.02 and the average experiment duration was 271 seconds.

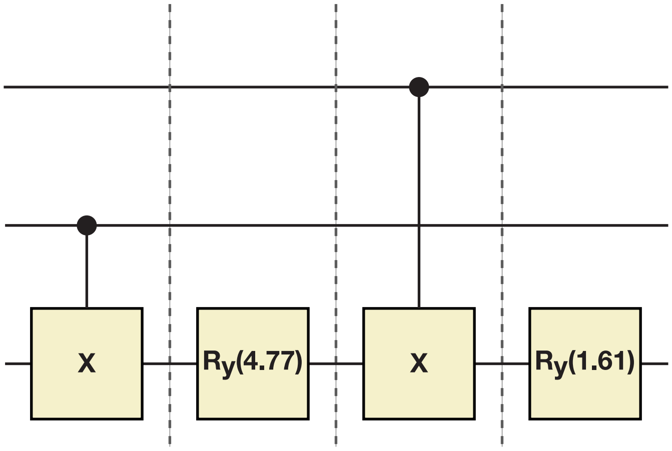


(6.36)

The best circuit over all experiments has a cost of 22.0 and is shown in 6.37 and 6.38.

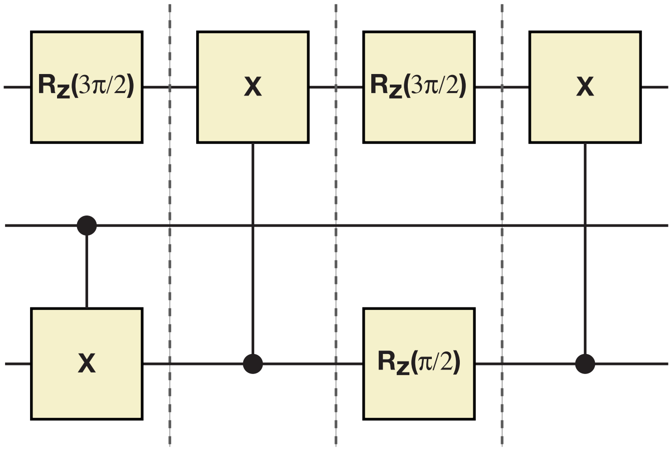
```
((((1) Id) ((2 3) CNOT))
(((1) Id) ((2) Id) ((3) Ry 4.77414915092167))
(((1 3) CNOT) ((2) Id))
(((1) Id) ((2) Id) ((3) Ry 1.60820981184512)))
```

(6.37)



(6.38)

This result is exactly the same as that which was obtained by Williams and Gray [66] and is less expensive than the original invention by Brassard, as shown in 6.39, which has a cost of 30.2.



(6.39)

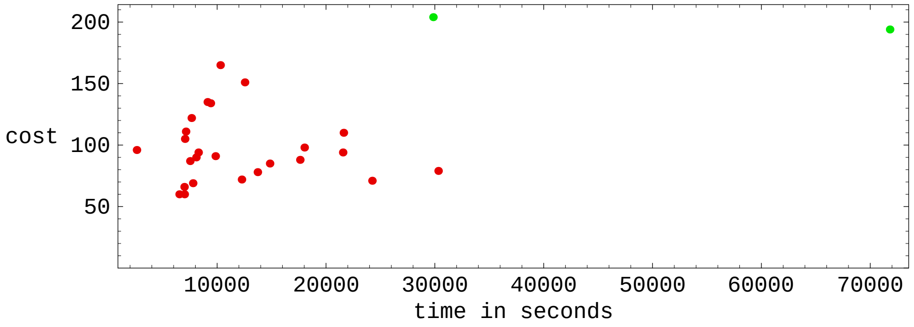
6.9 TOFFOLI

This experiment was performed with the experimental parameters as described in 6.40.

G =	**TOFFOLI**	
Elementary Gate Set =	parameterized universal	
Population Size =	1000	
Maximum Initial Length =	25	(6.40)
Maximum Dwell =	200	
Mutation Rate =	.02	
Cooling Rate =	.99	

Note: Since this circuit is believed to require six 2 qubit gates the *minimum cost* was set to 60.

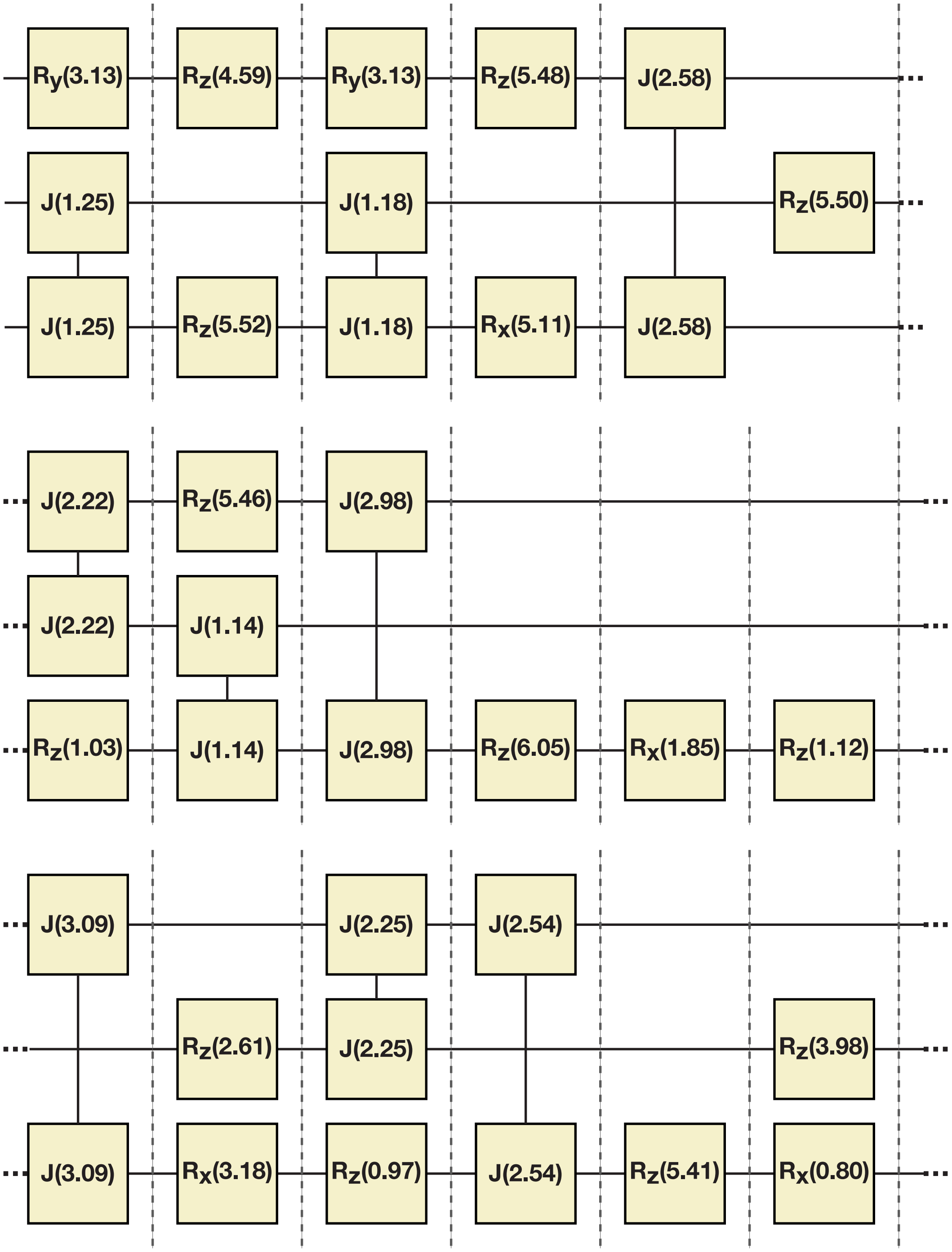
A total of 27 experiments were performed and the results are summarized in 6.41. The success rate was 0.07 and the average experiment duration was 14951 seconds.



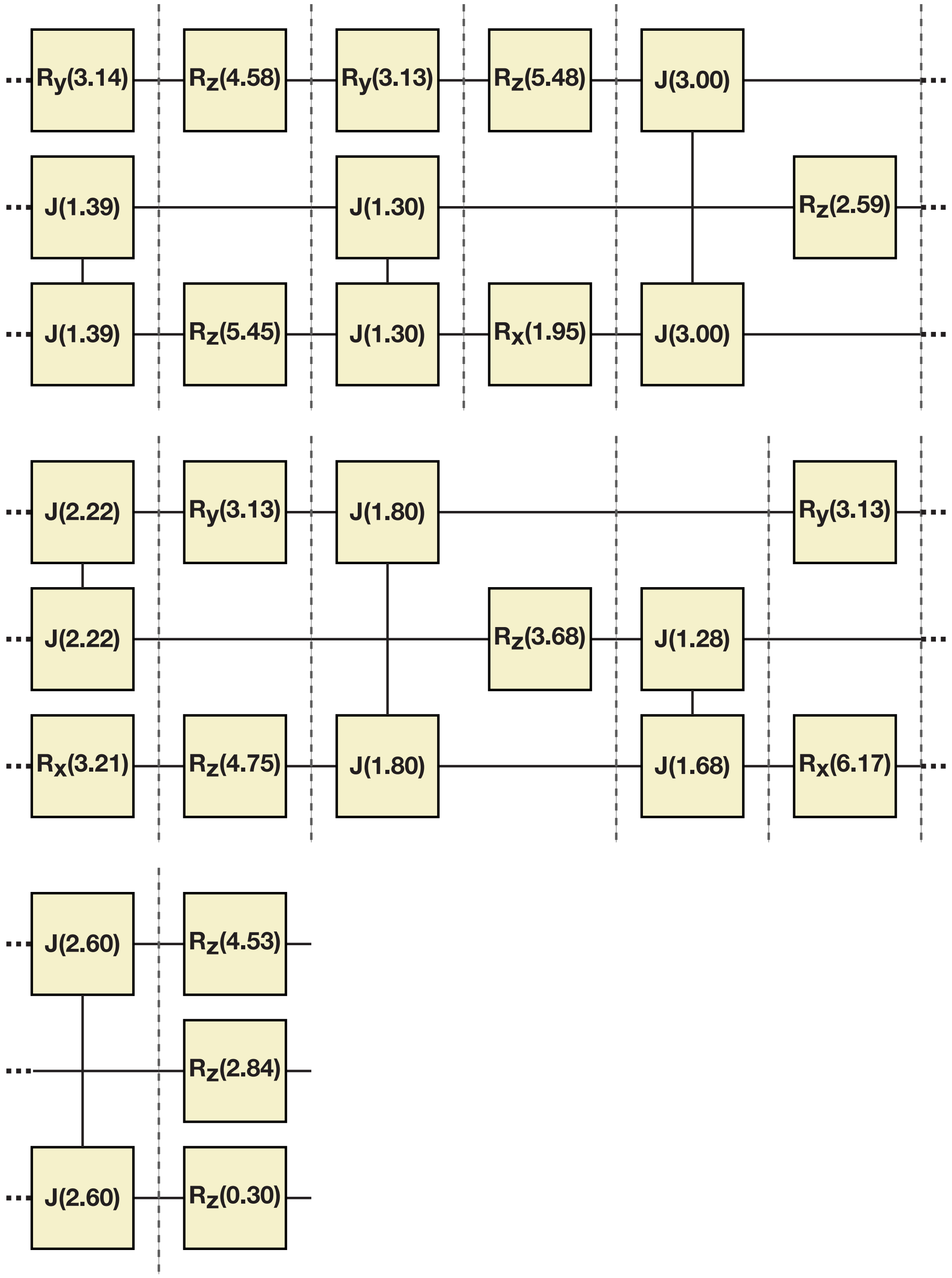
(6.41)

The best circuit over all experiments has a cost of 194.0 and is shown in 6.42 and 6.43.

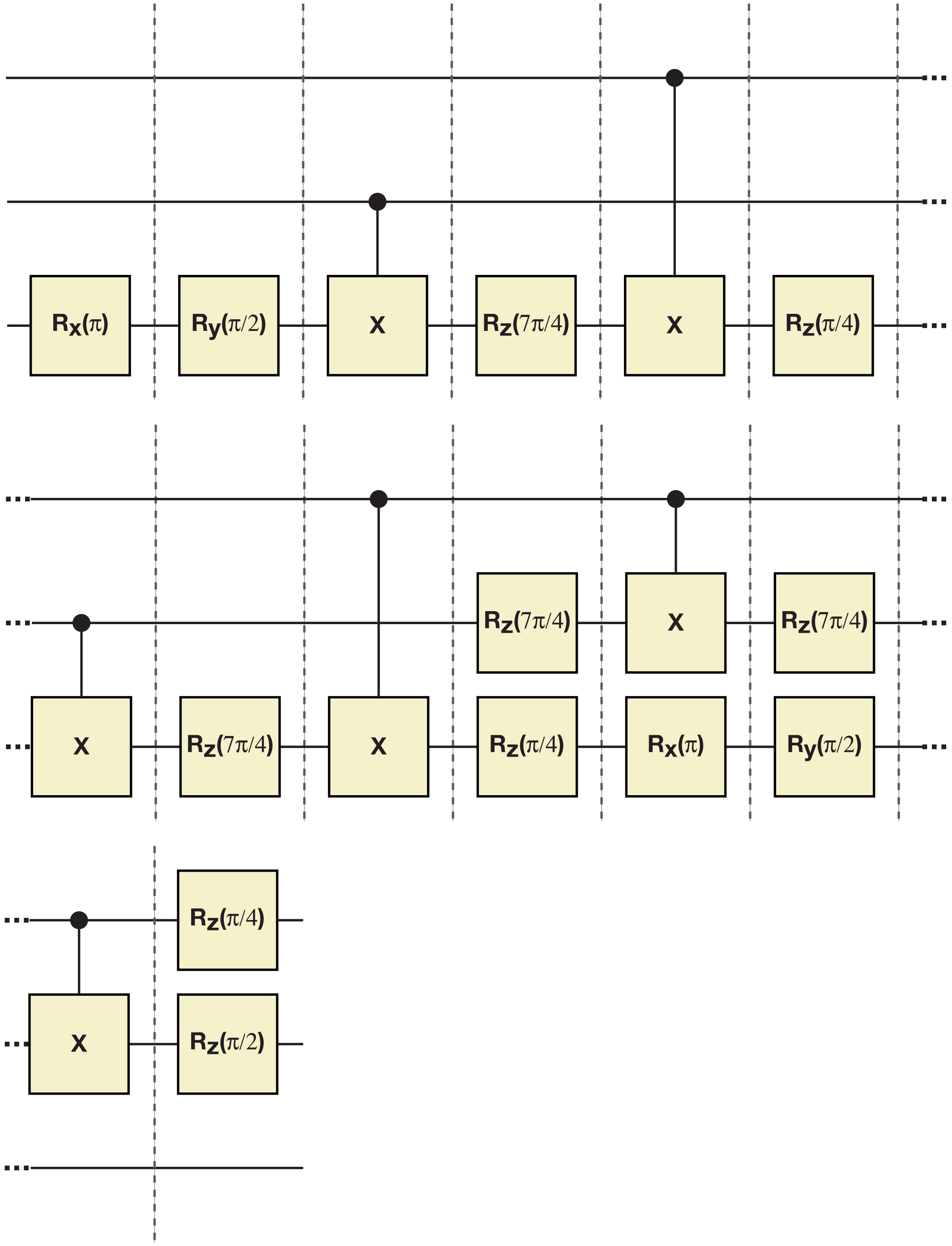
```
((((1) Ry 3.12967987957023) ((2 3) J 1.25464021954367))
(((1) Rz 4.58646779678745) ((2) Id) ((3) Rz 5.51543629905162))
(((1) Ry 3.12669986224446) ((2 3) J 1.17884461735261))
(((1) Rz 5.48217127021923) ((2) Id) ((3) Rx 5.10716987748223))
(((1 3) J 2.57612911903958) ((2) Rz 5.50194516093045))
(((1 2) J 2.21621622323029) ((3) Rz 1.03436155284308))
(((1) Rz 5.45744280505076) ((2 3) J 1.1448986896617))
(((1 3) J 2.98051013216298) ((2) Id))
(((1) Id) ((2) Id) ((3) Rz 6.05131697131327))
(((1) Id) ((2) Id) ((3) Rx 1.84559021089618))
(((1) Id) ((2) Id) ((3) Rz 1.11837092474942))
(((1 3) J 3.09169255103958) ((2) Id))
(((1) Id) ((2) Rz 2.60726474913167) ((3) Rx 3.18479495058295))
(((1 2) J 2.24501687259656) ((3) Rz 0.971893438567361))
(((1 3) J 2.54265748084369) ((2) Id))                                  (6.42)
(((1) Id) ((2) Id) ((3) Rz 5.40540031951163))
(((1) Id) ((2) Rz 3.98210072591016) ((3) Rx 0.801043779048809))
(((1) Ry 3.13580259904846) ((2 3) J 1.39167435673733))
(((1) Rz 4.57507383459652) ((2) Id) ((3) Rz 5.45062409266207))
(((1) Ry 3.13041969826293) ((2 3) J 1.30283513771296))
(((1) Rz 5.48094910349731) ((2) Id) ((3) Rx 1.94550725137229))
(((1 3) J 3.00429172210091) ((2) Rz 2.58963614336956))
(((1 2) J 2.2159439333902) ((3) Rx 3.21362344257537))
(((1) Ry 3.12630551259164) ((2) Id) ((3) Rz 4.74627777770301))
(((1 3) J 1.79832799648728) ((2) Rz 3.68099971660999))
(((1) Id) ((2 3) J 1.27786959439254))
(((1) Ry 3.12792100079302) ((2) Id) ((3) Rx 6.16507069205455))
(((1 3) J 2.60259886024991) ((2) Id))
(((1) Rz 4.53362753376377) ((2) Rz 2.83795135688103) ((3) Rz 0.29517)))
```


Ry(3.13)
Rz(4.59)
Ry(3.13)
Rz(5.48)
J(2.58)
J(1.25)
J(1.18)
Rz(5.50)
J(1.25)
Rz(5.52)
J(1.18)
Rx(5.11)
J(2.58)
J(2.22)
Rz(5.46)
J(2.98)
J(2.22)
J(1.14)
Rz(1.03)
J(1.14)
J(2.98)
Rz(6.05)
Rx(1.85)
Rz(1.12)
J(3.09)
J(2.25)
J(2.54)
Rz(2.61)
J(2.25)
Rz(3.98)
J(3.09)
Rx(3.18)
Rz(0.97)
J(2.54)
Rz(5.41)
Rx(0.80)

(6.43)



The commonly utilized circuit [47] is shown in 6.44. Using the CNOT circuit obtained in 6.6 this circuit has a cost of 76.2.



(6.44)

Chapter 7

Conclusion

This work has explored the Evolutionary Programming approach to quantum circuit design that is directly based upon the phenomenally successful study of evolutionary circuit design provided by Koza [33], as well as the successful preliminary application to quantum circuits performed by Williams and Gray [66] and Rubinstein [52]. The software associated with this work provides a general purpose experimental platform for the evolutionary design of quantum circuits, and develops the previous approaches in a number of important ways, including:

- A symbolic representation scheme for describing parallel quantum circuits constructed out of an arbitrary elementary gate set, including gates with one or more parameters.
- A highly optimized fitness assessment procedure which measures average case correctness with proper respect for global phase equivalence.
- A feasibility assessment procedure which disallows certain forms of quantum circuits, facilitating the characterization of particular quantum computers, and also the application of any available problem insight.
- A cost assessment procedure which measures the difficulty of implementing a quantum circuit on a particular quantum computing architecture.

- Local search mutations which can optimize the numerical parameters of a quantum circuit, and perform peephole simplification.

The experimental results consider significantly more difficult problems than previously studied, and show that the EP approach is capable of producing both correct and practical solutions to circuit design problems, even when little problem insight is available. These results will be useful for quantitative comparison with future approaches.

7.1 Future Directions

There are five main directions for future research which are expected to facilitate the application of the EP approach to larger and more complicated quantum circuit design problems:

7.1.1 Developmental Representation of Quantum Circuits

Early work by Spector and Stoffel [61][60] and recent work by Koza et al. [33] has explored the incorporation of the astonishingly complicated processes of developmental physiology into the evolutionary programming approach. In this approach, the genotype does not directly describe a single solution, but rather a recursive process to build a parametric class of solutions. In the case of quantum circuits, the developmental genotype could describe a process for building an efficient circuit which emulates a parametric gate, such as $\mathbf{R_x}(\theta)$, even out of a small discrete elementary gate set. These parametric circuits would be very useful, but would clearly require significant computational resources.

7.1.2 Worst Case Correctness Assessment

The correctness function used in this work is an average case measure, and the experiments show that it provides enough information for the algorithm to perform satisfactorily.

However, there are many distinct quantum gates which are very similar on average. For example, the **TOFFOLI** gate acts exactly like the identity on $\frac{3}{4}$ of its domain, and so there are a great many trivial quantum circuits which act very similar to the **TOFFOLI**. To properly distinguish between those gates which are highly similar on average requires a measurement of the worst case. Recent work by Gilchrist, Langford, and Nielsen [16] provides a highly informative discussion on the still open question as to how to best measure the distance between quantum gates, and they propose two techniques for worst case measurement. While these worst case techniques are significantly more complex than the average case used in this work, it seems highly likely that they would pay off by providing the algorithm with more power to distinguish between local and global optimums.

7.1.3 Fault Tolerance

Quantum error correction is perhaps the most successful aspect of quantum computing so far, but since one of the fundamental strengths of evolutionary programming is its ability to operate with minimal or no bias with regards to the design of solutions, it would be interesting to see if it could be used to discover new techniques. A simple Monte Carlo approach to this would be modify the genotype to phenotype transformation so that a noise step is inserted after each step in the circuit. The noise step simulates the effect of noise on the operation of the previous step. For example, a simple noise step could consist of random single qubit rotations applied to each qubit, where the angles are sampled from a normal distribution with zero mean and some small variance. Therefore each time a circuit is performed the outcome is slightly different, and its correctness would then be obtained as some statistical function over a sufficiently large number of samples.

7.1.4 Problem Insight

Any new insights into the local and global structure of quantum circuits may yield general heuristics which could be put to use to improve the quality of the results and reduce the required amount of computational resources. Work done by Lomont [38] has produced about a thousand useful quantum circuit simplification rules which could be used to greatly expand on the peephole simplification used in this work. Other work by Bullock, O'Leary, and Brennen [6] and Shends, Bullock, and Markov [56] provides insight into how and why some circuits are easier to construct than others. These insights could lead to a solid theoretical basis for feasibility conditions which could be used to significantly reduce the search space, and hence accelerate the search.

7.1.5 Larger Scale Experiments

The experimental results in this work were obtained in a relatively short period of time using a single small computer, and provide clear motivation for larger scale experiments using greater computational resources. The scalability of evolutionary programming has been established by Koza, who recently performed several massive experiments which employed the use of over 600 hours of a 1000 node cluster computer [31] [32]. The experiments resulted in six electronic circuits, all of which had only been first invented by human engineers and patented less than three years ago. These results indicate that evolutionary programming is a viable approach to large and difficult real world circuit design problems that is poised to enter the mainstream. Since quantum circuits are so much less understood than electronic circuits, it seems quite likely that the application of even moderately greater computational resources will yield several new and interesting quantum circuits.

Bibliography

[1] O. Aaserud and I. Ring Nielsen. Trends in current analog design: A panel debate. *Analog Integrated Circuits and Signal-Processing*, 7(1):5–9, 1995.

[2] D. Aharonov and M. Ben-Or. Fault-tolerant quantum computation with constant error. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 176–188, 1997.

[3] Robert Axelrod. *The Evolution of Cooperation.* Basic Books, 1984.

[4] P. Benioff. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *J. Stat. Phys.*, 22(5):563–591, 1980.

[5] G. Brassard. Teleportation as a quantum computation. In T. Toffoli, M. Biafore, and J. Leao, editors, *Proceedings of the Fourth Workshop on Physics and Computation (PhysComp '96)*, pages 48–50. New England Complex Systems Institute, 1996.

[6] Stephen S. Bullock, Dianne P. O'Leary, and Gavin K. Brennen. Asymptotically optimal quantum circuits for d-level systems. ArXiv e-print, Cornell University, 14 October 2004. http://www.arxiv.org/abs/quant-ph/0410116.

[7] A.W. Burks, H.H. Goldstein, and John von Neumann. Preliminary discussion of the logical design of an electronic computing instrument. Technical report, Institute for Advanced Study, Princeton, N.J, June 1946.

[8] S. Cook. The complexity of theorem proving procedures. In *Proc. 3rd ACM Symp. on the Theory of Computing*, pages 151–158, 1971.

[9] Charles Darwin. *The Origin of Species by Means of Natural Selection: Or, the Preservation of Favored Races in the Struggle for Life.* John Murray, London, 1859.

[10] Richard Dawkins. *The Selfish Gene*, chapter 11. Memes: the new replicators. Oxford University Press, 1976,1989.

[11] D. Deutsch. Quantum computational networks. In *Proceedings of the Royal Society of London Ser. A*, volume 425, pages 73–90, 1989.

[12] David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. In *Proceedings of the Royal Society of London Ser. A*, volume 400, pages 97–117, 1985.

[13] R. P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6/7):467–488, 1982.

[14] G. B. Fogel and D. W. Corne, editors. *Evolutionary Computation in Bioinformatics.* Morgan Kaufmann, San Francisco, CA, 2003.

[15] Edward Fredkin and Tommaso Toffoli. Conservative logic. *Int. J. Theor. Phys.*, 21:219–253, 1982.

[16] A. Gilchrist, N. K. Langford, and M. A. Nielsen. Distance measures to compare real and ideal quantum processes. LANL quant-ph archives, 10 August 2004. http://xxx.lanl.gov/abs/quant-ph/?0408063.

[17] Kurt Gödel. On formally undecidable propositions of principia mathematica and related systems. *Anzeiger der Akad. d. Wiss. in Wien (math.-naturw. Kl.)*, 19, 1930.

[18] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings, 28th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 212–219, May 1996.

[19] Jin-Kao Hao, Frédéric Lardeux, and Frédéric Saubion. Evolutionary computing for the satisfiability problem. In *Applications of Evolutionary Computing*, volume 2611 of *LNCS*, pages 259–268, University of Essex, England, UK, 14-16 April 2003.

[20] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[21] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[22] D.A. Huffman. A method for the construction of minimum-redundancy codes. In *Proceedings of the I.R.E.*, pages 1098–1102, September 1952.

[23] R. Karp. *Complexity of Computer Computations*, chapter Reducibility among Combinatorial Problems, pages 85–103. Plenum Press, New York, 1972.

[24] R. Kelsey, W. Clinger, and J. Rees (eds.). Revised5 report on the algorithmic language scheme. *Higher-Order and Symbolic Computation*, 11(1), August 1998.

[25] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 13 May 1983.

[26] E. Knill, R. Laflamme, and W. Zurek. Threshold accuracy for quantum computation. (online preprint quantph /9610011), LANL, 1996.

[27] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, 1992.

[28] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge, MA, 1994.

[29] John R. Koza. *Engineering Design Synthesis*, chapter : Automatic synthesis of both the topology and numerical parameters for complex structures using genetic programming, pages 319–337. Springer, London, 2003. Amaresh Chakrabarti ed.

[30] John R. Koza, Forrest H. Bennett III, David Andree, and Martin A. Keene. *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann, San Francisco, CA, 1999.

[31] John R. Koza, Lee W. Jones, Martin A. Keane, Matthew J. Streeter, and Sameer H. Al-Sakran. *Genetic Programming Theory and Practice II*, chapter 8. Toward automated design of industrial-strength analog circuits by means of genetic programming, pages 121–142. Kluwer Academic Publishers, 2004. Una-May O'Reilly, Rick L. Riolo, Gwoing Yu and William Worzel eds.

[32] John R. Koza, Martin A. Keane, and Matthew J. Streeter. Routine high-return human-competitive evolvable hardware. In Ricardo S. Zebulum, David Gwaltney, Gregory Hornby, Jason Lohn Didier Keymeulen, and Adrian Stoica, editors, *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware*, pages 3–17, Los Alamos, CA, 2004. IEEE Computer Society Press.

[33] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence.* Kluwer Academic Publishers, 2003.

[34] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*, chapter 15. Automated Reinvention of Six Post-2000 Patented Circuits, pages 421–482. Kluwer Academic Publishers, 2003.

[35] R Laflamme. personal correspondence.

[36] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Soft.*, 5:308–323, 1979.

[37] H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406:974–978, 2000.

[38] Chris Lomont. Quantum circuit identities. ArXiv e-print, Cornell University, 16 July 2003. http://arxiv.org/abs/quant-ph/0307111.

[39] Y. Manin. Computible and uncomputible (in Russian). Sovetskoye Radio, Moscow, 1980.

[40] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1968.

[41] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*, chapter 2. Introduction to quantum mechanics, pages 60–119. Cambridge University Press, 2000.

[42] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*, chapter 3. Introduction to computer science, pages 120–170. Cambridge University Press, 2000.

[43] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*, chapter 5. The quantum Fourier transform and its applications, pages 216–247. Cambridge University Press, 2000.

[44] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*, chapter 4. Quantum circuits, pages 171–215. Cambridge University Press, 2000.

[45] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*, chapter 1. Quantum Computation, page 23. Cambridge University Press, 2000.

[46] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*, chapter 5. The quantum Fourier transform and its applications, page 219. Cambridge University Press, 2000.

[47] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*, chapter 4. Controlled operations, page 183. Cambridge University Press, 2000.

[48] Bill O'Neill. Digital evolution. *PLoS Biology*, 1(1):11–14, October 2003.

[49] G. Papadopoulos and G. Wiggins. A genetic algorithm for the generation of jazz melodies. In *Proceedings of the Finnish Conference on Artificial Intelligence (SteP'98)*, Jyvaskyla, Finland, 7-9 September 1998.

[50] John Preskill. Reliable quantum computers. *Proc.Roy.Soc.Lond.*, A454:385–410, 1998.

[51] Thomas Quarles, A. R. Newton, D. O. Pederson, and A. Sangiovanni-Vincentelli. *SPICE 3 Version 3F5 User's Manual.* Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, March 1994.

[52] B. Rubinstein. Evolving quantum circuits using genetic programming. In J. R. Koza, editor, *Genetic Algorithms and Genetic Programming at Stanford 2000*, pages 325–334. Stanford Bookstore, Stanford University, CA, 2000.

[53] R.W.Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 26(2):147–160, April 1948.

[54] C.E. Shannon. A symbolic analysis of relay and switching circuits. *Transactions of the AIEE*, 57:713–723, 1938.

[55] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[56] Vivek V. Shende, Stephen S. Bullock, and Igor L. Markov. Recognizing small-circuit structure in two-qubit operators and timing hamiltonians to compute controlled-not gates. ArXiv e-print, Cornell University, 2003. http://www.arxiv.org/abs/quant-ph/0308045.

[57] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings, 35th Annual Symposium on Foundations of Computer Science.* IEEE Press, 1994.

[58] K. Sims. Artificial evolution for computer graphics. In *Computer Graphics (Siggraph '91 Proceedings)*, pages 319–328, July 1991.

[59] K. Sims. Evolving virtual creatures. In *Computer Graphics (Siggraph '94 Proceedings)*, pages 15–22, July 1994.

[60] Lee Spector and Kilian Stoffel. Automatic generation of adaptive programs. In Pattie Maes, Maja J. Mataric, Jean-Arcady Meyer, Jordan Pollack, and Stewart W. Wilson, editors, *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior: From Animals to Animats 4*, pages 476–483. MIT Press, 1996.

[61] Lee Spector and Kilian Stoffel. Ontogenetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 394–399. MIT Press, 28–31 July 1996.

[62] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proc. London Math. Soc, Ser. 2*, 42(1936-7):230–265, 1936.

[63] A. M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

[64] Lieven M. K. Vandersypen, Costantino S. Yannoni, and Isaac L. Chuang. Liquid state nmr quantum computing. *Encyclopedia of Nuclear Magnetic Resonance*, 9:687–697, 2002.

[65] William Whitney. Darwinian rhapsody. *Psychology Today Magazine*, Mar/Apr 2004.

[66] Colin P. Williams and Alexander G. Gray. Automated design of quantum circuits. In *Selected papers from the First NASA International Conference on Quantum Computing and Quantum Communications (QCQC'98)*, pages 113–125, 17-20 February 1998.

[67] Felix L. Winkelmann. *CHICKEN User's Manual*. http://www.call-with-current-continuation.org/.

Appendix A

Quantum Gate Definitions

$$\mathbf{R_x}(\theta) = \begin{pmatrix} \cos(\frac{\theta}{2}) & -\mathbb{i}\sin(\frac{\theta}{2}) \\ -\mathbb{i}\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix}$$

$$\mathbf{R_y}(\theta) = \begin{pmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ -\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix}$$

$$\mathbf{R_z}(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & \mathbb{e}^{\mathbb{i}\theta} \end{pmatrix}$$

$$\mathbf{X_1} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\mathbf{Y_1} = \begin{pmatrix} 0 & -\mathbb{i} \\ \mathbb{i} & 0 \end{pmatrix}$$

$$\mathbf{J}(\theta) = \begin{pmatrix} \mathbb{e}^{-\mathbb{i}\theta} & 0 & 0 & 0 \\ 0 & \mathbb{e}^{\mathbb{i}\theta} & 0 & 0 \\ 0 & 0 & \mathbb{e}^{\mathbb{i}\theta} & 0 \\ 0 & 0 & 0 & \mathbb{e}^{-\mathbb{i}\theta} \end{pmatrix}$$

$$\mathbf{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\mathbf{CPHASE} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \mathbb{i} \end{pmatrix}$$

$$\mathbf{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{QFT2} = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \mathbb{i} & -1 & -\mathbb{i} \\ 1 & -1 & 1 & -1 \\ 1 & -\mathbb{i} & -1 & \mathbb{i} \end{pmatrix}$$

$$\mathbf{SQRTSWAP} = \frac{1}{2}\begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 1+\mathbb{i} & 1-\mathbb{i} & 0 \\ 0 & 1-\mathbb{i} & 1+\mathbb{i} & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

$$\textbf{TELEPORTSEND} = \frac{1}{2}\begin{pmatrix} 1 & 0 & -1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & -1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & -1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & -1 & 0 & -1 & 1 & 0 & -1 & 0 \\ -1 & 0 & -1 & 0 & 0 & 1 & 0 & -1 \end{pmatrix}$$

$$\textbf{TELEPORTRECEIVE} = \frac{1}{2}\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\textbf{TOFFOLI} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$