
Chapter 90

Evolving Quantum Programs and Protocols

Susan Stepney, John A. Clark

Department of Computer Science, University of York, Heslington,
York, YO10 5DD, UK

Contents

1	Introduction.....	2
1.1	Traditional Computing.....	2
1.2	Quantum Computation.....	3
1.3	Engineering the Physics and the Applications.....	3
2	Quantum Mechanics	4
2.1	A Photonic Introduction	4
2.2	Qubits, States and Operations on them.....	6
2.3	Making Measurements.....	10
2.4	The Strange Case of Entanglement.....	11
3	Quantum Algorithms	12
3.1	Classical System Development and the Drive to Abstraction	12
3.2	The Circuit Model of Quantum Computation	12
3.3	Choice of Gate Set.....	13
3.4	Particular Algorithms.....	14
4	Evolutionary Computation.....	19
4.1	Introduction.....	19
4.2	Search terminology	19
4.3	Biology	21
4.4	Evolutionary algorithms in general	23
4.5	Evolutionary algorithms in particular	25
4.6	Genetic Programming.....	26
4.7	Variations on a theme	28
5	Evolving Quantum Algorithms: implementation issues.....	29
5.1	Representation of Potential Solutions.....	29
5.2	Spector’s Push-Based System.....	33
5.3	Evaluating Candidate Solutions: Cost and Fitness Functions	34
5.4	Structure of the search landscape	36
5.5	Hand Processing	37

5.6	Simulation Issues	37
6	Evolving Quantum Algorithms: results	38
6.1	Circuits for Classic Combinatorial Problems	38
6.2	Deterministic to Probabilistic: Massey <i>et al.</i>	41
6.3	Hitting the Physics: How many pulses does it take to make a <i>CN</i> ? ..	44
6.4	Starting in the Right Place	46
6.5	Communication, Teleportation and Entanglement	46
6.6	Visualisation	51
6.7	Summary	52
7	Conclusions	53
8	The Future	54
8.1	Improving the simulation efficiency	54
8.2	New areas to explore	54
8.3	Other applications	55
9	Bibliography	56

1 Introduction

1.1 Traditional Computing

The modern computer is one of the greatest technical developments of the 20th Century. It pervades most aspects of our lives. In a very short space of time we have moved from computers that filled many rooms to Weiser's ubiquitous computing vision [95], where computers disappear into the fabric of everyday objects (pens, cups, spectacles etc.) and much computation goes on behind the scenes.

The performance of modern computing platforms has grown at a significant rate; a home computer is barely out of the box before a newer, more powerful one is being promoted by its manufacturers. Gordon Moore (co-founder of Intel) observed in 1965 that the number of transistors per unit area in an integrated circuit had doubled each year. Subsequently, a slower but still impressive rate of doubling approximately every 18 months has been observed. (This is generally referred to now as Moore's Law.) Early home computers had only a few kilobytes of memory. Today, commercially available 'memory sticks', measuring perhaps ½ inch by ¼ inch by 1 inch, can store a gigabyte or more of information.

Such performance gains cannot continue indefinitely. Indeed, as circuitry gets ever smaller, with components approaching the atomic scale, the laws of physics will present barriers to how much further this technology can be pushed. However, if the laws of physics present a practical problem, for some applications the laws of physics also provide a radical solution.

1.2 Quantum Computation

If computers are one of the greatest practical developments of the 20th Century, then quantum mechanics must stand as one of its greatest intellectual achievements. Despite the controversy that has marked its development, it is now beyond question that, as model of behaviour of small-scale phenomena, quantum mechanics, as we understand it, ‘works’. Although the ‘meaning’ of quantum mechanics is still hotly debated, down on the ground scientists freely use its mathematical theory as given. But only recently, the *consequences* of the basic mathematical theory were recognised as being deeper than we had thought.

In a keynote speech in 1981, Nobel Laureate Richard Feynman noted that harnessing quantum phenomena of matter could allow complex systems to be simulated effectively [32]. In particular, he proposed that this could be a way of simulating various quantum mechanical systems. Paul Benioff was actively researching quantum computation around this time [7] [8].

In 1985 David Deutsch showed how the classical computation model (the Turing machine) could be simulated using quantum mechanical properties of matter [26]. Since the Turing Machine is felt to capture what is meant by ‘computation’ (see the discussion in [70, Chapter 1]), the whole of classical computation could, in principle, be carried out using quantum mechanics. Subsequent developments showed that the laws of physics could be used to achieve results faster than could be achieved using classical computing. Deutsch showed the first ‘faster than classical’ computation [26] (concerning the single bit XOR or parity problem). This was later generalised by Deutsch & Jozsa to distinguishing between balanced and constant functions on n Boolean variables [27].

The biggest practical impetus came in 1994 when Peter Shor demonstrated a quantum analogue of the Discrete Fourier Transform [83]. This could be harnessed effectively to perform factorisation. Most importantly, a product $n = pq$ of two large primes could be factorised highly efficiently (‘in polynomial time’). If factorisation can be carried out efficiently then swathes of public key cryptography are broken and so much communications is rendered insecure. Factorisation had become quantum computing’s ‘killer application’. We simply do not know whether efficient factorisation is possible by classical means, but if it is, we seem far from achieving it.

1.3 Engineering the Physics and the Applications

Quantum computation is not yet with us in any practical sense (the largest number of qubits currently in a quantum computer is 7), but a significant body of physical scientists are at work on making quantum computing a practical reality [10]. If history repeats itself, we will get small computers initially that will grow as technology improves. Since quantum computers seem capable of achieving results unachievable by other means, exploiting effectively even limited hardware platforms may bring significant economic benefits.

We now have an opportunity to build the application infrastructure to run on quantum computers when they eventually come on-stream. Several researchers have developed new and important quantum algorithms over the past decade (see section 3.4) but there are fundamentally few distinct quantum algorithms. In some ways novel application development seems to have stalled, as Williams & Clearwater note:

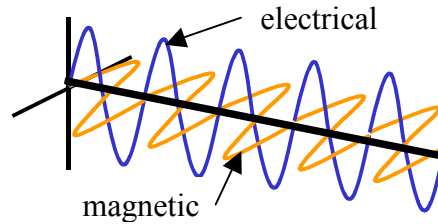


Figure 1. Light as a transverse electromagnetic wave.

Of course computer scientists would like to develop a repertoire of quantum algorithms that can, in principle, solve significant computational problems faster than any classical algorithm. Unfortunately the discovery of Shor's algorithm for factoring large composite integers was not followed by a wave of new quantum algorithms for lots of other problems. To date, there are only about seven quantum algorithms known. — [96]

Why is this? The authors of this review believe that intuition about quantum phenomena and the nature of quantum computation is too limited. It is such a radically different arena, well outside the comfort zone provided by traditional computation. If our mindsets are the problem then we must seek to free ourselves, or augment our current capabilities. Nature, in the guise of quantum mechanical laws, provides us with new computational capabilities. But Nature also is good at invention; evolution is a form of continual reinvention. In the rest of this chapter, we review how automated search techniques inspired by biological systems can be used to uncover new quantum circuits and algorithms.

2 Quantum Mechanics

First we provide the background needed to understand this approach, starting with an overview of quantum computation and the mathematics needed to support it.

2.1 A Photonic Introduction

To illustrate some interesting quantum mechanical phenomena we consider the propagation and filtering of white light.¹ Light is a transverse electromagnetic wave. Photons have electrical and magnetic field components (oscillating in planes perpendicular to each other, and perpendicular to the direction of light propagation) as shown in Figure 1. The electrical plane of vibration is shown as vertical, but in practice this may be any plane perpendicular to the direction of propagation (with a corresponding change in magnetic plane of vibration).

Suppose we have a white light source, and a target, represented by an eye (Figure 2a). Consider light travelling from the source to the target. This light comprises a vast number of photons, each with its own plane of polarisation (hereafter referred to as its polarisation). The polarisation of emitted photons may be regarded as uniformly distributed, that is, all possible polarisations are equally likely.

¹ Similar explanations can be found in [70] [96] [77].

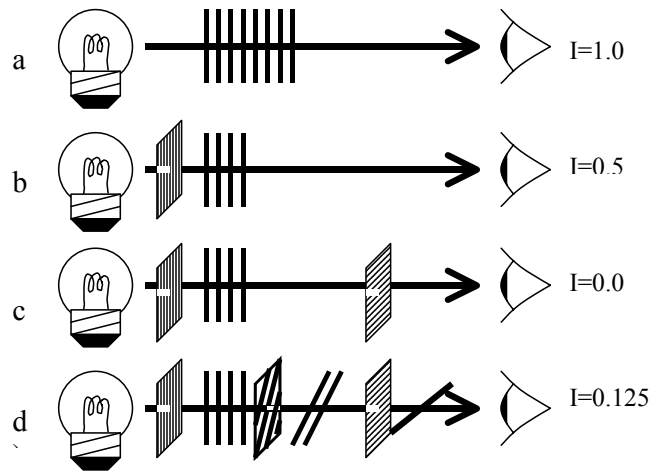


Figure 2. Polarisation experiments

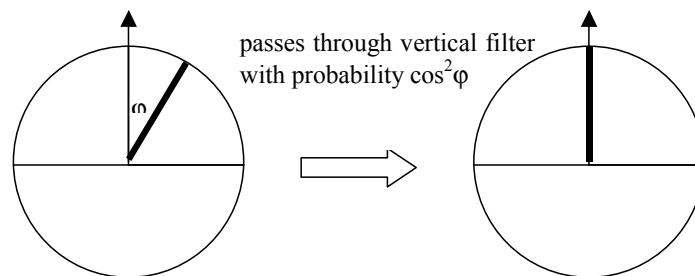


Figure 3. Photon passes through vertical filter

Suppose now we put a vertically polarising filter between the source and the target (Figure 2b). Vertically polarised photons pass through the filter and horizontally polarised photons will be absorbed by it. But what about photons with an intermediate polarisation? 50% are absorbed by the filter and 50% pass through the filter and emerge the other side as vertically polarised. Now add a horizontally polarising filter, (Figure 2c). No photons pass through this filter.

If a *diagonally* polarising filter is now placed between the other two, light now appears at the target, with 12.5% of the intensity of the original (Figure 2d). Oddly, adding a filter which impedes light (only half the photons incident on it will pass through) now allows light to reach the target where previously it did not. What is happening?

Whether or not a photon is absorbed is not determined by its initial polarisation; rather its *chances* of being absorbed are determined by its polarisation. The probability of a photon passing through a filter depends only on the angle ϕ made by its polarisation with the direction of the polarising filter. This probability is given by $\cos^2 \phi$. The probability of being absorbed is $\sin^2 \phi$. This is shown in Figure 3.

We may feel that photons with identical polarisation should behave identically, yet it appears that the ‘decision’ to pass through or be absorbed is made only at the time of incidence. Let us denote a vertically polarised photon by $|0\rangle$ and an absorbed photon by $|1\rangle$. We can think of a photon with polarisation at angle to the vertical as being some ‘mixture’ of horizontal and vertical and denote its state by $|\psi\rangle = (\cos \phi)|0\rangle + (\sin \phi)|1\rangle$. Incidence with the filter can be thought of as a challenge to the photon. It must probabilistically decide which state, $|0\rangle$ or $|1\rangle$, it ‘wants’ to be. We refer to being forced to choose in this way as a *measurement*.

Measurement clearly affects the state of the photon. Some schemes use the fact that measurement affects quantum state as the basis of cryptographic key distribution schemes.

We can now understand the behaviour exhibited in the filtering experiment described above. If we average out over all angles ϕ , a randomly selected photon is equally likely to be absorbed by the vertical filter or to emerge from it, vertically polarised. The horizontal filter blocks with probability 1 all photons emerging vertically polarised from the vertical filter (since the angle ϕ is 90 degrees). However, when vertically polarised photons are incident on a diagonally polarised filter, they have a 50% chance of passing through and emerging diagonally polarised. Thus, we would expect 25% of photons to pass successfully through both vertical and diagonal filters. Finally a diagonally polarised photon has a 50% chance of passing through the horizontally polarised filter. Thus we would now expect 12.5% of photons to make it through all three filters.

Each filter provided a measurement, collapsing the photon into one of two orthogonal states. The experimenter determines which states could arise (by choosing the polarisation of the filter). Different filters define different orthogonal output states and so provide different measurements.

2.2 Qubits, States and Operations on them

A standard computational memory bit can be in one of two stable states: 0 or 1. A corresponding two-state quantum bit, or *qubit* for short, may similarly be in one of two computational ‘basis states’, which we shall denote by $|0\rangle$ and $|1\rangle$. But quantum bits can also exist in a complex *superposition* of these states. A superposition $|\Phi\rangle$ is denoted by $|\Phi\rangle = a|0\rangle + b|1\rangle$ where the coefficients a and b are complex numbers *normalised* such that $|a|^2 + |b|^2 = 1$. We can represent such a complex superposition in vector form:

$$|\Phi\rangle = a|0\rangle + b|1\rangle = \begin{pmatrix} a \\ b \end{pmatrix}$$

When measured with respect to the standard basis the probability of being measured as a $|0\rangle$ is $|a|^2$ and the probability of being measured as a $|1\rangle$ is $|b|^2$. The basis states can be seen as extremes of superpositions (that is, $a = 1, b = 0$ denotes $|0\rangle$ and $a = 0, b = 1$ denotes $|1\rangle$). The vector is referred to as the system’s *state amplitude vector*, or just *state*.

Classical computing uses both reversible and non-reversible computing elements. The classical operation NOT is reversible, since $\text{NOT}(\text{NOT } A) = A$, whether A is true or false, but the classical operation AND is not reversible, since $(A \text{ AND } B) = \text{false}$ loses the information about the sources A and B : there are three possible source combinations. In quantum computing, the fundamental operations are *physically reversible transformations*. Whatever is done, can be undone. Such operations are *unitary transformations* and represented by *unitary matrices*. An n by n matrix U is unitary if and only if

$$U^{T*}U = UU^{T*} = I_n$$

where T denotes the transpose operation (that is, $u^T_{ij} = u_{ji}$) and $*$ denotes that each element of the matrix has been replaced with its complex conjugate.² An operation on a state is represented by the application of the corresponding unitary transformation to that state vector. For example, the matrix N below represents the NOT operation and we can see that $N|0\rangle = |1\rangle$ and $N|1\rangle = |0\rangle$.

$$|0\rangle \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$N = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad N \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad N \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

A particularly useful single-qubit operation is the Hadamard transformation H :

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

$$H \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

$$H \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

Applying H to $|0\rangle$ (the second line of the above series of formulae) results in an equal superposition of the basis states. We see later why this transform is so useful. There are many single qubit gates in addition to the NOT gate and Hadamard gate. Other well-known single-qubit transformations are given by the Pauli spin matrices:

$$\sigma_0 = I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \sigma_x = X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

$$\sigma_y = Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \sigma_z = Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

There is a generally applicable decomposition for single-qubit unitary transformations:

$$U = e^{i\alpha} \begin{pmatrix} e^{-i\phi} & 0 \\ 0 & e^{i\phi} \end{pmatrix} \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} e^{-i\psi} & 0 \\ 0 & e^{-i\psi} \end{pmatrix}$$

Application of matrices is *linear*. Thus, if our state vector is in a genuine complex superposition, we have:

$$U(a|0\rangle + b|1\rangle) = aU|0\rangle + bU|1\rangle$$

² For a complex number $z = x + yi$, the complex conjugate is $\bar{z} = x - yi$

This captures a crucial property of quantum mechanics, that it is thought to be linear. Indeed, if it is not linear, issues such as communication back in time may become possible [74].

For realistic computations we need to operate on more than one qubit. A system comprising n qubits has 2^n possible basis states, denoted $|0..0\rangle .. |1..1\rangle$. A complex superposition can be represented in vector form similar to the above. Thus, for a two-qubit system, a complex superposition is represented as:

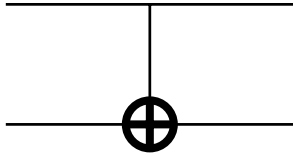
$$|\Phi\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

where the coefficients are normalised: $|a|^2 + |b|^2 + |c|^2 + |d|^2 = 1$. A unitary transformation on this system is represented by a 4 by 4 unitary matrix.

An important operation on two qubits is the *CN* (Conditional NOT) operation, defined by

$$CN|00\rangle = |00\rangle, \quad CN|01\rangle = |01\rangle, \quad CN|10\rangle = |11\rangle, \quad CN|11\rangle = |10\rangle$$

For each basis state this operation flips the value of the second qubit if and only if the first qubit has value 1. It implements a form of logical XOR transformation $CN|xy\rangle = |x, x \oplus y\rangle$. The *CN* matrix and diagrammatic representations are given below:

$$CN = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$


(See section 3.2 for an explanation of the diagrammatic representation.)

Operations on a single qubit may affect every probability amplitude element in the state vector. For example, in a two-qubit system, applying a NOT operation to the first qubit should carry out the following transformation

$$a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle \xrightarrow{N} c|00\rangle + d|01\rangle + a|10\rangle + b|11\rangle$$

But the NOT operation is defined over a single qubit, by the 2 by 2 matrix N . To apply it to one qubit in a 2-qubit system, we need some way of ‘composing’ this operation with the identity operation on the second qubit. This composing is carried out by forming the *tensor product* of the two appropriate matrices:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & 1 \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ 1 \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & 0 \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} c \\ d \\ a \\ b \end{pmatrix}$$

This serves to ‘lift’ the single-qubit NOT operation to the whole system. We can similarly compose more complex unitary operations on subsystems, for example, we can lift a 4 by 4 CN matrix acting on qubits 3 and 4 to act on a 4-qubit system (producing a 16 by 16 matrix). (When an operation acts on non-adjacent qubits some implementation tricks are needed.)

We have already been using shorthand for of tensor notation. Basis states such as $|01\rangle$ are actually tensor products:

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \times \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ 0 \times \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Applying the n -fold product of Hadamard matrices for each qubit to a basis state $|000\dots 0\rangle$ gives a complex uniform superposition:

$$H^{\otimes n} |00\dots 0\rangle = H_1 \otimes H_2 \otimes \dots \otimes H_n |00\dots 0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle$$

All possible basis states are equally likely to be observed. Now consider the $n+1$ qubit state $|00\dots 00\rangle|0\rangle$. Apply the above n -fold Hadamard product to the first n qubits (composing with the identity for the final qubit):

$$H^{\otimes n} \otimes I_{n+1} |00\dots 00\rangle|0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle|0\rangle$$

If U_f is a unitary transformation that acts on a basis state $|x\rangle|0\rangle$ to produce $|x\rangle|f(x)\rangle$, then by linearity we obtain

$$U_f \sum_{x=0}^{2^n-1} |x\rangle|0\rangle = \sum_{x=0}^{2^n-1} U_f |x\rangle|0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle|f(x)\rangle$$

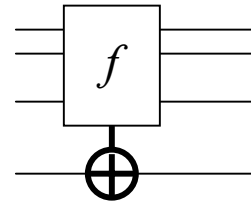
Thus superposition and linearity can combine to allow all values of $f(x)$ to be calculated simultaneously. Note that the operation U_f is entirely reversible. We have kept the index values x and so the classical function f need not itself be reversible. A more general approach, where the value of the target qubit is not necessarily 0, is to use unitary functions U_f defined by

$$U_f |x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle$$

The above shows how the mathematics of quantum mechanics implies the potential for massive computational parallelism but eventually, we will want to observe some ‘result’ of the system, we will wish to make *measurements* of the system. Very little comes for free, and it is in the area of measurement that quantum mechanics exacts a price for the free parallelism it provides, as we shall see later.

In some problems we will be presented with an unknown binary function f (an *oracle function*) and will seek to determine some property of it (for example, if it is balanced or constant). Such functions can be represented by the unitary transform that acts on m input qubit values q_1, \dots, q_m and a single output qubit, by flipping the output qubit value whenever $f(q_1, \dots, q_m)$ is true. The unitary transformation and diagrammatic representation are:

$$|q_1, \dots, q_m, q_{out}\rangle \rightarrow |q_1, \dots, q_m, f(q_1, \dots, q_m) \oplus q_{out}\rangle$$



(See section 3.2 for an explanation of the diagrammatic representation.)

2.3 Making Measurements

Free from external interference, quantum systems evolve according to the famous wave equation of Erwin Schrödinger (see discussion in [96]). However, observation of a qubit ‘forces its hand’, losing all uncertainty. It is measured as either a $|0\rangle$ or a $|1\rangle$, and any global states inconsistent with this observation are rendered impossible to subsequent observation. This is what is usually referred to in the Copenhagen interpretation³ of quantum mechanics as ‘state space collapse’. Consider the case of

$$|\Phi\rangle = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle$$

Suppose we observe the value of the second qubit. We are equally likely to measure a $|0\rangle$ as a $|1\rangle$. Suppose we measure a value of $|1\rangle$. The state now ‘collapses’ to a state consistent with this observation:

$$|\Phi'\rangle = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

The probability amplitudes of two of the basis states are now zero (and those states do not appear in the above formula). The amplitudes of the other two basis states have been normalised so that we have a valid final state superposition. If we now observe the value of the first qubit we are equally likely to observe a value of $|0\rangle$ (and the state becomes $|01\rangle$) as to observe a value of $|1\rangle$ (and the state becomes $|11\rangle$).

We can see, informally, how superposition, linearity and state space collapse *might* provide a useful means of answering some computational questions. Suppose

³ There are other interpretations of quantum mechanics, the most important of these for quantum computation being Everett’s ‘many worlds’ interpretation [31]. The interested reader is referred to [29] [28] for more information. For present purposes we eschew philosophical discourse and largely just present the mathematics. It is widely held that the mathematics ‘works’.

we have a function $f(x)$ defined over $0 \dots 2^n - 1$. As indicated above we create a complex superposition of the basis states and apply the corresponding unitary transformation U_f to obtain:

$$U_f \sum_{x=0}^{2^n-1} |x\rangle |0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle |f(x)\rangle$$

Suppose that we wish to find an x such that $f(x)$ takes on a specific value, say 1. If we could arrange, in some way, to observe a $|1\rangle$ for the $(n+1)$ th qubit, we would have achieved our aim. This is because all subsequent observations on the index component qubits (the first n qubits) must be consistent with this observation, and so we must see qubit values representing a value of x for which $f(x) = 1$. How one arranges to make highly convenient observations (in this case observing a $|1\rangle$ for the value of the result qubit) forms the crux of many a quantum algorithm. We typically avail ourselves of massive parallelism afforded by superposition and linearity, and seek to ‘bump up’ the probability amplitudes of those basis states that form the answers we want.

2.4 The Strange Case of Entanglement

Entanglement is one of the strangest phenomena in physics. It is also a computational resource. Consider the following superposition of two qubits:

$$|\Phi\rangle = \frac{1}{\sqrt{2}} |00\rangle + \frac{1}{\sqrt{2}} |11\rangle$$

Suppose we measure the first qubit. We are equal likely to observe a $|0\rangle$ or a $|1\rangle$. If we measure a $|0\rangle$ then the system collapses to $|00\rangle$. If we now measure the second qubit we will observe $|0\rangle$ with probability 1. If we initially measure a $|1\rangle$ for qubit 1 then the system collapses to $|11\rangle$. If we now measure the second qubit we will observe $|1\rangle$ with probability 1. So observation of the first qubit fully determines the state of the second qubit. Suppose qubit 1 is in York in England, and qubit 2 is in New York in the USA. A measurement taken in York affects the measurement taken in New York! This is sometimes referred to as ‘spooky action at a distance’. It was not initially believed by scientists, but has been confirmed by experiment.

The above qubits are said to be *entangled*. Two qubits are said to be *entangled* if their global state cannot be expressed as a tensor product of single qubit states. Our two-qubit system cannot be expressed in the form of a tensor product, that is, in the form

$$\frac{1}{\sqrt{2}} |00\rangle + \frac{1}{\sqrt{2}} |11\rangle = (a|0\rangle + b|1\rangle) \otimes (c|0\rangle + d|1\rangle) = ac|00\rangle + ad|01\rangle + bc|10\rangle + bd|11\rangle$$

since no choice of a, b, c, d can make the above identity hold.

3 Quantum Algorithms

We now examine some issues in quantum algorithms, their representation, and development.

3.1 Classical System Development and the Drive to Abstraction

Abstraction is a necessary component of controlling complexity, and it comes in many forms in classical computing. A brief scan of the history of development of programming languages, for example, reveals an increasing trend towards abstraction, and higher level constructs. Programmers initially wrote only object code, the lowest level of instruction for computer hardware. This was largely superseded by assembly languages, which allow more complex operations to be carried out by instructions and allowed locations to be given names. Higher-level languages such as Fortran and C emerged, and modern programming languages, such as the object-oriented Java, allow complex operations to be performed with single programming instructions.

Software engineering progress has gone hand-in-hand with the ability to address concepts at higher levels of abstraction. Classical software development has a lifecycle, from the elicitation of requirements for the system, through progressive refinement into lower level designs, down to code that can be compiled for a hardware platform. Many of these steps enjoy some degree of automation: high level programs are transformed to object code by compilers; code itself can be generated automatically from designs expressed in certain mathematical or diagrammatic notations.

The software industry has become adept at this refinement process; we know how to get from problem to solution, though this may require many thousands of person-years effort for the largest systems. In particular, software engineering has learned from both its mistakes and its successes, and has codified this learning in the form of *patterns*, reusable chunks of expertise. Patterns were originally developed in the domain of architecture by Christopher Alexander [1], and have been enthusiastically adopted in the software engineering community. Patterns can occur at any stage of the development lifecycle, for example [90] [34] [35] [6].

In the quantum domain there is a good deal of apparatus at the low level, in terms of the operation of unitary gates, but not much at higher levels of abstraction. We have nothing like the classical software engineering intellectual infrastructure in the quantum domain, yet it needs to be developed if quantum computing is to achieve its full promise [91].

3.2 The Circuit Model of Quantum Computation

A common representation of a quantum algorithm is the quantum circuit diagram. It bears some similarity to a combinational logic circuit diagram. The evolution of a qubit state is represented by its path from left to right along a horizontal ‘wire’. As the qubit moves along the wire it may meet a ‘gate’ that acts on its state. Gates are encountered in the order they are applied. Gates may act on one, two or more qubits. If a qubit is not acted on by a gate, there is an implicit tensor product with the identity transform on that qubit. Figure 4 shows a simple circuit model.

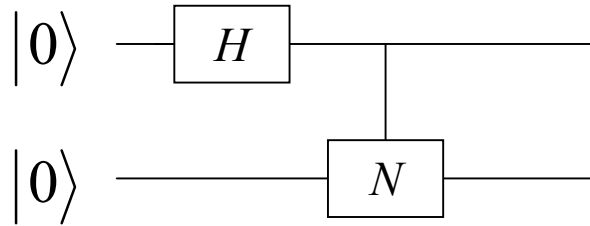


Figure 4. A circuit model of a simple algorithm

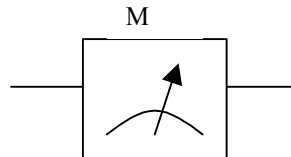


Figure 5. A measurement gate M

Initially the system is in the state $|00\rangle$. As we move from left to right we first apply the Hadamard transform H to the first qubit, and then apply a controlled-NOT with the first qubit as control and the second as the target. Thus, the evolution is given by

$$|00\rangle \xrightarrow{H} \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right)|0\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle \xrightarrow{CN} \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

In some cases it is possible to express the final state as a tensor product, and so meaningfully assign states to individual qubits. For example, after the application of just the Hadamard gate, the state of the system is a tensor product (as shown in the equation above). However, the final superposition is *not* expressible as a tensor product; we can talk only about the state of the whole system, not the states of individual qubits.

We have already seen a two-qubit gate: the controlled-NOT, CN . In general, *any* unitary transform gate can give rise to a controlled variant involving an additional control qubit. Thus, the Pauli matrix X gives rise to a controlled- X gate on two qubits. Since this controlled gate itself implements a unitary transform, it too can be lifted to a controlled variant on three qubits, a controlled-controlled- X gate (where the gate X is applied to the target qubit if and only if the two control qubits have value $|1\rangle$), and so on.

We can choose to measure the value of a qubit using a measurement gate. This is how information gets out of the system. We will observe either a $|0\rangle$ or a $|1\rangle$. Figure 5 shows a measurement gate acting on a qubit.

3.3 Choice of Gate Set

Given the plethora of possible quantum gates, what sets of gates should a designer use when considering algorithm development? The gate set must involve at least one multiple qubit gate (since otherwise all operations would result in states that are expressible as tensor products). The gate set must be logically sufficiently powerful

to allow arbitrary algorithms to be implemented⁴ but must be guided also by the practicalities of realisation.

Simple gates will map fairly directly onto physical operations on a small number of qubits. Complex gates implementing complex transformations will need to be broken down into a series of simpler physically achievable gates. We know how to compose the operation of a series of transformations: simply lift each simple operation to the whole system and multiply the matrices for each such lifted operation. However, we lack a systematic means of factoring complex operations into a series of smaller convenient operations. Convenience here may be affected by various criteria: ease of implementation of the smaller transformations; speed of execution of smaller gates and their composition etc. Note also that even a simple operation of controlled-NOT may be more easily implemented if the two qubits concerned are physically close to each other. Different operations may differ in their ease of implementation according to the underlying architecture mechanisms used, for example, quantum dots will favour the implementation of a different gate set than would NMR based quantum computing.

3.4 Particular Algorithms

There are in fact very few distinct quantum algorithms. Indeed, it was this observation that prompted the authors to engage in using search techniques to look for quantum algorithms in the first place. Below we outline some of the most important algorithms.

3.4.1 Deutsch-Josza Promise

Quantum computation seems particularly suited to problems where some ‘global property’ is sought. The first such algorithm to demonstrate faster than classical behaviour was the Deutsch-Josza promise algorithm [27]. Suppose a black box calculates the value of a Boolean function $f(x)$ over a range $x = 0 \dots 2^n - 1$, and there is a guarantee, or *promise*, that the function is either constant ($f(x) = 0$, or $f(x) = 1$, for all x) or is balanced (equal numbers of input values x give $f(x) = 0$ as give $f(x) = 1$). How much effort is required to determine whether the function is constant or balanced?

For the simplest case of x in the range 0..1, we must carry out two classical function evaluations. But in the quantum case we need only one. Since this initially seems such a counter-intuitive result, we describe the working of the Deutsch-Josza algorithm in some detail.

Start with the state $|0\rangle|1\rangle = |01\rangle$. Now apply the Hadamard transformation to the first and then to the second qubit, to give

$$|\Psi\rangle = \frac{1}{2}|0\rangle(|0\rangle - |1\rangle) + \frac{1}{2}|1\rangle(|0\rangle - |1\rangle)$$

Now apply the unitary function U_f defined by

$$U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle$$

⁴ This may be challenged, though the general motivation is clearly sensible. Williams & Gray [97] note “An incomplete gate set may make sense when the properties of the target computation allow it.”

to give

$$\begin{aligned} U_f |\Psi\rangle &= \frac{1}{2} |0\rangle (|f(0)\rangle - |1 \oplus f(0)\rangle) + \frac{1}{2} |1\rangle (|f(1)\rangle - |1 \oplus f(1)\rangle) \\ &= \frac{1}{2} (-1)^{f(0)} |0\rangle (|0\rangle - |1\rangle) + (-1)^{f(1)} |1\rangle (|0\rangle - |1\rangle) \end{aligned}$$

Now we have

$$U_f |\Psi\rangle = \begin{cases} \pm \frac{1}{2} (|0\rangle + |1\rangle) (|0\rangle - |1\rangle), & \text{if } f(0) = f(1) \\ \pm \frac{1}{2} (|0\rangle - |1\rangle) (|0\rangle - |1\rangle), & \text{if } f(0) \neq f(1) \end{cases}$$

So the first qubit takes each of two orthogonal values depending on whether the function is balanced or not. Now apply the Hadamard transformation to the first qubit, to obtain

$$|\Phi\rangle = \pm \frac{1}{\sqrt{2}} (|f(0)\rangle \oplus |f(1)\rangle) (|0\rangle - |1\rangle)$$

So by measuring the value of the first qubit (one measurement) we can determine with certainty whether the function is balanced.

Although we are able to determine whether the function is constant or balanced with a single measurement, we cannot characterise the exact function. If the function is constant we cannot tell whether it is $f(0) = f(1) = 0$ or $f(0) = f(1) = 1$. We have given up specific information on values of $f(x)$ for a global property of all such values.

The above algorithm can be extended to work equally efficiently on n variables.

Calculating global properties efficiently seems to be a task to which quantum computation is well suited. The promise problem is a very restricted one, with little practical application but its solution is theoretically important. The exploitation of quantum phenomena for global property elicitation seems a promising avenue for further work, both for quantum algorithm development by theorists and evolutionary search advocates.

3.4.2 Grover's Algorithm – Searching an Unstructured Database

Consider a function f on the domain $0 \dots 2^n - 1$ with a single value v in this domain such that some predicate $p(v) = \text{true}$. Can we find this index value v ? In classical computing our best attempt is enumeration, which on average takes 2^{n-1} tests. Full enumeration takes 2^n tests. Grover, however, demonstrates how a quantum search of $O(\sqrt{2^n})$ is possible [40].

Although the various papers talk about unstructured 'database' search, the principal applications are those for which the database values are calculated. First place the system in a superposition

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle$$

Now apply the operator U_v , defined such that it inverts the amplitude if the predicate is satisfied:

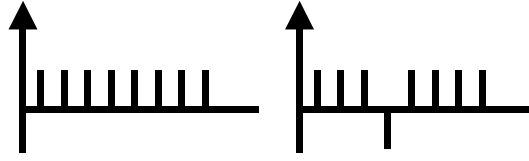


Figure 6. Amplitude negation for the correct index value

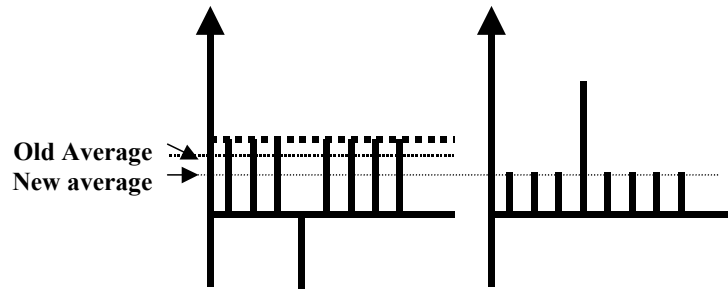


Figure 7. Inversion about the average

$$U_v |x\rangle = \begin{cases} -|v\rangle, & x = v \\ |x\rangle, & x \neq v \end{cases}$$

It is possible also to apply an operator that inverts about the average:

$$\begin{pmatrix} -1 + \frac{2}{2^N} & \frac{2}{2^N} & \cdots & \frac{2}{2^N} \\ \frac{2}{2^N} & -1 + \frac{2}{2^N} & \cdots & \frac{2}{2^N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{2}{2^N} & \frac{2}{2^N} & \cdots & -1 + \frac{2}{2^N} \end{pmatrix}$$

The application of these two operators in succession is shown in Figure 6 and Figure 7. Inverting the amplitude of the identified element (here $x = 3$) reduces the average amplitude. When we invert about this new average the amplitude of the identified element is increased (but all the others are decreased). By repeating this process we can further increase the amplitude of the identified element (and so increase our chances of observing this element). We omit the details here, but the description provides a rough motivation for the algorithm.

There are enhancements of this algorithm. The original algorithm gave a 50% chance of seeing the right result. Subsequent developments have produced more reliable variants. Also, the approach can be extended to cater for several index states satisfying the predicate of interest. If there are R such ‘marked states’ the algorithm will deliver one such state in a search of order $O(\sqrt{2^n / R})$. (The original single marked state algorithm has $R = 1$.) However, the procedure can easily be overcooked; perform too many iterations and the amplitudes of interest will start to

decrease in magnitude⁵. The optimal number of iterations depends on the number R of marked states. (This may not be known but quantum state counting algorithms have been developed. See [70].) A summary of Grover's algorithm can be found in [59].

Grover's search can be regarded as the quantum analogue of brute force enumeration. It does not avail itself of structure in a particular problem (this is what is meant by the term 'unstructured database').

3.4.3 Shor's Quantum Discrete Fourier Transform and Hidden Subgroups

In 1994 Peter Shor's Quantum Discrete Fourier Transform (QDFT) [83] gave quantum computing its 'killer application': composite number factorisation. Shor showed how the QDFT could be used to determine the periodicity of given function in polynomial time. A result from number theory shows how obtaining the period of a particular function can allow composite numbers to be factorised. Hence the QDFT gives a polynomial-time factorisation algorithm on a quantum computer. Much public key cryptographic security depends on the supposed computational difficulty of factorisation. We do not give the details of the algorithm here: suffice it to say that the algorithm bumps up probabilities of states that are some multiple of periods part.

The QDFT has applications to other problems, for example phase estimation and order finding. The general *hidden subgroup problem* remains a significant focus of interest. (See [70, Chapter 5].)

3.4.4 Teleportation

Teleportation uses properties of quantum mechanics to transport precisely a qubit state from one location to another. A simple teleportation circuit is shown in Figure 8.

The first two gates (Hadamard and controlled-NOT) place the second and third qubits in a *maximally entangled* state (described in section 3.2). This is done in advance of preparing the source qubit $|\Psi\rangle$. The second qubit is sent to Alice and the third qubit (now entangled with the second) is sent to Bob. Now suppose the qubit state we Alice wants to transmit is

$$|\Psi\rangle = a|0\rangle + b|1\rangle$$

After the next two operations (controlled-NOT and Hadamard) the state of the system is

$$|\Phi\rangle = \frac{1}{2}|00\rangle(a|0\rangle + b|1\rangle) + \frac{1}{2}|01\rangle(a|1\rangle + b|0\rangle) + \frac{1}{2}|10\rangle(a|0\rangle - b|1\rangle) + \frac{1}{2}|11\rangle(a|1\rangle - b|0\rangle)$$

In each of the four state components the third qubit's state is defined by a simple transformation of that of the original source qubit. If Alice measures the values of the first two qubits the state reduces to a normalised form of one of the four components. If Alice informs Bob of the measurement results (M1 and M2) Bob can use this information to apply a suitable inverse transformation to his third entangled qubit to recover the value of the first source qubit.

⁵ The reader is invited to verify this informally.

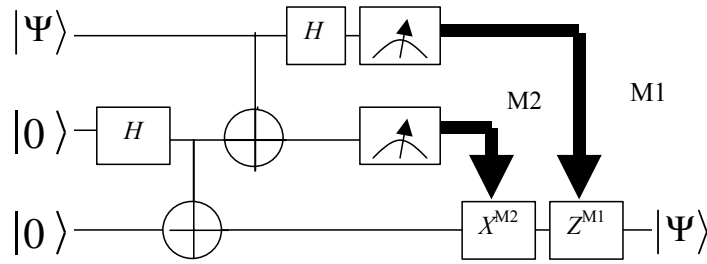


Figure 8. Teleportation circuit

For example, if Bob is informed that two measured values were $|01\rangle$ he can deduce that the remaining state is

$$|\Phi\rangle = |01\rangle(a|1\rangle + b|0\rangle)$$

By applying the transformation X to the his (third) qubit Bob can recreate the initial Ψ since

$$X \begin{pmatrix} b \\ a \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} b \\ a \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix}$$

Similarly if Alice measures $|11\rangle$ and communicates the results to Bob, Bob can apply X followed by Z to recover the initial qubit, since

$$ZX \begin{pmatrix} b \\ a \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} -b \\ a \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} q \\ -b \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix}$$

The general solution is to apply X^{M2} followed by Z^{M1} , where X^{M2} means apply X if $M2 = 1$ and do nothing otherwise (that is, apply the identity) etc.

Note that the original state in Alice's source qubit is lost. It has been spirited away to Bob's qubit. No information has been created.⁶ A state infinitely rich in information (any values of a and b may be used) has been teleported across to another place at the expense of sending only two classical bits of information. However, we cannot extract all this information by measurement; when measured we will see a $|0\rangle$ or a $|1\rangle$.

Teleportation shows us the power of entanglement as a resource. We have presented it here as an algorithm, but we might just as easily consider it a basic gate. It all depends on what level of abstraction we wish to use.

The teleportation circuit provides an analogue of a well known classical state swapping algorithm. Suppose x and y are locations containing bit values. A traditional approach to swapping involves the use of a temporary location z . The program is:

$$z := x; \quad x := y; \quad y := z;$$

⁶ It is a feature of quantum information that it cannot be copied. This is the celebrated 'no-cloning' theorem.

A more efficient solution that uses no temporary location, and the XOR function, is:

$$x := x \oplus y; \quad y := x \oplus y; \quad x := x \oplus y;$$

Starting with the classical circuit for this program, Mermin carries out justified replacements to derive a quantum analogue [66], which is the teleportation circuit above. This work is intriguing since it suggests the possibility of a more systematic approach to finding quantum analogues of classical circuits.

4 Evolutionary Computation

4.1 Introduction

We will use *search* as a way to explore the space of quantum circuits. The search space is large, even with only a handful of gate types, and a handful of qubits. So we need an *effective* algorithm to search this very large space; an effective algorithm will necessarily sample only a very small part of the search space, yet must find good solutions. We concentrate on the metaheuristic search technique of *Evolutionary Algorithms*, inspired by the biological process of evolution. Before we apply this to quantum circuits, we give some background on search and evolutionary algorithms in general.

Michalewicz & Fogel [67] provide an excellent introduction to a range of modern heuristic search techniques.

4.2 Search terminology

4.2.1 Solution space and objective function

The *solution space* Σ is the space comprising the real world artefacts of interest, such as electrical circuits, antenna designs, building plans, computer programs, musical tunes, and so on, that we wish to search for optimal members. In our case Σ is the space of quantum circuits.

The *objective function* ϕ measures the real world property to be optimised, such as efficiency, accuracy, path length, speed, power consumption, and so on. ϕ maps each element of the solution space to a real number that expresses how ‘good’ it is, in terms of the real world property. So $\phi : \Sigma \rightarrow \mathfrak{R}$. This objective may be difficult to capture or quantify: for example, how might we rank melodious music?

If there are multiple objectives (such as simultaneous high speed and low power consumption), ϕ can be generalised into an objective vector and used to pursue multi-objective optimisation, or the objectives can be suitably weighted and combined in a single ϕ . From now on, we assume a scalar objective function.

4.2.2 Search space

Before optimal solutions can be searched for, they need to be encoded into some computer representation convenient for search. The chosen representation forms the *search space* S .

The choice of search space is an important modelling decision. It should both fit the problem naturally, and be searchable by the chosen algorithm. The search space

may be closely related to the solution space (for example, a simple numerical representation of certain parameters of interest), or it may be a less direct representation (for example, a computer program that, when executed, generates the solution space element). That is, the *representation function* Γ , that maps the search space to the solution space, $\Gamma: S \rightarrow \Sigma$, may be simple, or extremely complicated.

The simplest, and possibly most common, choice of search space is bit strings of length l , $S = \{0,1\}^l$, that directly encode the parameter values being optimised as a binary value. More structured strings of integers and characters can be used, for example, encoding the component values in a fixed topology electronic circuit. The search space can comprise finite state machines, for example, as predictors of the next value in a sequence. And the search space can comprise full computer programs, for example, ones that on execution draw a variable topology electronic circuit diagram. Here, execution of the program can be thought of as application of the representation function Γ .

A change of representation can ‘smooth’ the search landscape, or make it more searchable in other ways. For example, a numerical parameter can be encoded as a bit string using conventional binary coding or Gray coding. With binary coding, changes to high bits has a bigger effect than changes to low bits, whereas with Gray coding consecutive underlying numbers differ by only one bit flip. So Gray coding gives a much smoother, more continuous, search landscape, but it may smooth out important features

Other changes of representation are possible. A change of basis (for example, by rotating to use the eigenvectors as the basis) can make structure clearer. A standard data transformation (Fourier, Laplace, and so on) might make the space more searchable by highlighting the key properties. Projecting onto a lower dimensional space loses some information: if that information is not relevant the space becomes much smaller and more searchable. Alternatively, embedding in a higher dimensional space can smooth the search space. Ultimately, indirect encodings as programs that generate results allow the maximum flexibility.

4.2.3 The fitness landscape

The *evaluation function* f evaluates each element of the search space, $f: S \rightarrow \mathfrak{R}$. This function should be in a form suitable for efficient computation, and for use by the chosen search algorithm.

Clearly, f should also be correlated with the objective function: that is, optimising the fitness should simultaneously optimise the objective. It is common merely to take $f = \Gamma \circ \phi$, effectively ignoring the distinction. In the case of very indirect encodings, this may be necessary, as may be no other useful relationship between the search and solution spaces. But this particular choice is not necessary in general, and transforming the evaluation function in some suitable way can dramatically alter the efficiency of the search: the choice of evaluation function is as much a modelling decision as the choice of search space representation. Furthermore, it is not even necessary to require strict simultaneous optimisation, provided that the optima of f give ‘good enough’ answers when transformed into the solution space, or give answers suitable as the starting points for more refined searches.

f is usually called the *fitness function* if it is being maximised, and the *cost function* if it is being minimised, although terminology is not consistent. Some search implementations require f to be positive.

The fitness function is often described as defining a *fitness landscape* over the search space, by analogy to the way height information describes a topographical landscape over physical space in the world. Thinking of the fitness function in these terms, it becomes natural to talk of ‘peaks’ of fitness by analogy with mountain peaks, and of ‘hill climbing’ as a way of ascending to the peaks. A *local optimum* is then any peak, and the *global optimum* is the highest peak in the landscape, the fitness Everest. (When using cost functions, the terminology is of ‘valleys’.) The analogy holds most closely when the search space is a 2-dimensional space of real numbers. In practice, the search space is more often bit strings or computer programs, and the analogy becomes more strained, since the space no longer has the continuity or topology of the original.

4.2.4 Search algorithm

The task of the search algorithm is to find the global optimum, or, more usually, a ‘sufficiently good’ local optimum.

There are two main classes of search algorithms: solitary and population based. Solitary algorithms (such as hill climbing, and simulated annealing) consider a single search point s at each step, and generate a trace, or *trajectory*, of (search point, evaluation result) pairs $T^{(t)} = \langle (s^{(0)}, r^{(0)}), \dots (s^{(t)}, r^{(t)}) \rangle$. Population-based algorithms (such as evolutionary algorithms, and swarm algorithms) consider a set of search points s_i at each step, and generate a *trajectory* of sets of (search point, evaluation result) pairs $T^{(t)} = \langle (s^{(0)}, r^{(0)})_i, \dots (s^{(t)}, r^{(t)})_i \rangle$.

The meat of the search algorithm is its *move function*, which determines which part of the space to sample next, given the results from the already sampled space, $M: T \rightarrow S$. Commonly, the move function is memoryless; it is a function of only the current state $(s^{(t)}, r^{(t)})$, and not of the entire trajectory. Less commonly it takes into account some information about earlier states (for example, tabu searches), and less commonly still, the entire trajectory of the search so far.

The algorithm also needs a starting point, $s^{(0)}$. This is often a random start state, or may be seeded with “good” known solutions, especially in hybrid searching combining several algorithms.

4.3 Biology

Simple search algorithms tend to get trapped on local optima, since the entire local neighbourhood comprises worse solutions, and it is difficult to know how to find a better one. Evolutionary algorithms employ a population of candidate solutions to explore the search space, they use small variations (‘mutations’) to generate new candidate solutions, and some combine solutions from different places in the landscape (different ‘parents’); all these provide a means to escape from local optima.

This section describes (in *very* simplified terms) the biology of evolution. Nearly every statement made here about biological evolution is not absolutely true: there are always peculiar organisms that do something slightly out of the ordinary, and great subtleties in the actual mechanisms. However, the descriptions given here are the ideas that have provided the inspirations for evolutionary algorithms.

4.3.1 Inheritance + variation + selection = evolution

We can consider a reproducing population as attempting to solve some optimisation problem, of exploring some search space.

Consider any reproducing population that exhibits the three characteristics of **inheritance** (offspring resemble their parents, so good solutions are preserved), **variation** (offspring are not identical to their parents, so the population does not get ‘frozen in’ to a poor solution), and **selection** (there is preferential survival of those that best meet the search criterion, so that good solutions survive in preference to poor ones).

Any such population ‘improves’ with respect to the problem it is solving : it evolves. Note that this argument is independent of any specific mechanisms for inheritance, variation, or selection.

4.3.2 Early pioneers of biological evolution

Charles Darwin’s grandfather, Erasmus Darwin (1731–1802), had some early ideas on evolution: “the strongest and most active animal should propagate the species, which should thence become improved” [23].

Jean-Baptiste Lamarck (1744–1829) proposed a mechanism by which inheritance occurs: that traits acquired due to environmental effects can be passed to the next generation. For example, a blacksmith develops strong arms from wielding a hammer, and this results in stronger sons; or fish in dark environments do not use their eyes, and so subsequent generations have vestigial eyes. We now know that this is not the mechanism that is actually used in biological evolution, but the idea can be adapted for use in artificial evolution in some circumstances.

Charles Darwin (1809–1882) promulgated his famous theory of “descent with modification”, involving the key steps of inheritance (with no mechanism proposed), variation, and natural selection (“survival of the fittest”) [22].

4.3.3 DNA – a mechanism for inheritance

DNA, the ‘double helix’ deoxyribonucleic acid molecule, provides (part of) the mechanism for biological inheritance. DNA provides the ‘instructions’ for ‘building’ an organism. More accurately, it provides the instructions for building various protein molecules, which in turn are key in the development of the organism, but the environmental context also plays a key role.

DNA is passed from parent to offspring, which provides the inheritance. It also provides variation, because the DNA can mutate, and (in the case of sexual reproduction) is inherited from two different parents.

4.3.4 Genotype v. phenotype

The *genotype* of the organism is its complement of DNA, and is what is inherited.

The *phenotype* of the organism is its physical form, which develops from the original single egg cell (containing its DNA and many other necessary chemicals) by a process of *morphogenesis*. This is an incredibly complicated, highly non-linear process, and can be likened to a ‘one-way function’: it is not computationally feasible to deduce the genotype from the phenotype.

Environmental effects as hammer-wielding or dark adaptation affect the phenotype, not the genotype, and so are not inherited. So the Lamarckian mechanism is not, and can not be, the correct mechanism in biological evolution.

4.3.5 Endogenous and exogenous fitness

Fitness, the biological survival probability, is also a function of the phenotype. It is organisms, not DNA, that reproduce, survive, and die. In the case of biology, the

fitness function is ability to survive and reproduce, provided endogenously by the system (and its definition can therefore sound somewhat circular).

In artificial evolutionary algorithms, the fitness function is provided exogenously, by the designer. The artificial population's survival depends on this externally determined fitness.

4.4 Evolutionary algorithms in general

This biological process provides the metaphor for evolutionary search algorithms (EAs). The most general form of an EA has the following pattern:

```
initialise population ;  
while not stopped  
    evaluate population ;  
    select parents of next generation ;  
    breed next generation ;  
return fittest in population ;
```

The specific algorithms incorporate a multitude of variations and optimisations around this theme.

4.4.1 Search and solution space representations

The biological genotype corresponds to the search space representation, and the phenotype to the solution space representation. The genotype is in the simplest case a string (usually called a 'chromosome'). Each element of the string (usually called an 'allele') tends to be a binary bit, but can also be an integer, a real number, a character, or any other data type appropriate for encoding the search space.

In EAs, the search and solution space representations tend to be very close, even identical. For example, in a direct encoding of parameter values of interest, the genotype might be the bit string "00101010", representing the phenotype integer "42".

However, there is the possibility to have a richer genotype/phenotype relationship. For example, interesting work is being done on certain encodings of genotypes and then have a 'developmental' phase that 'grows' the corresponding phenotype [55]. This is also the case of some forms of genetic programming, where the genotype is the program being evolved, and the phenotype is the result of execution of that program. Here the genotype tends to be some kind of tree representation of the program, and the phenotype can anything computable.

4.4.2 Initial population

It is conventional to initialise the population with random chromosomes. It is possible to seed this population with known 'good' but sub-optimal solutions. However, this may sometimes bias the search away from much better but very different solutions. Populations tend to contain on the order of 100–1000 individuals, but much smaller populations are also used.

4.4.3 Evaluation

For simple representations with small genotype/phenotype distance, the population of chromosomes can be evaluated directly, in terms of the appropriate fitness function.

In the case where there is a large distance between the genotype and phenotype, for example in the case of evolving computer programs, it is usually necessary to calculate the fitness in terms of the solution space objective function, as there is no simpler way of evaluating it. So every generation the population individuals have to be ‘grown’ (the genotype program is executed).

For genotypes that are computer programs intended to run on a range of inputs, the fitness is evaluated on a sample of all possible inputs.

In some cases, it is not possible to define an algorithm for the objective function, particularly in cases where evaluation involves a component of aesthetic appreciation, for example when evolving music or other artworks. In such a case the selection can be done manually, by presenting a selection of phenotypes to a human user, and asking them to choose the best, or rank the selection. Dawkin’s biomorphs, evolving ‘interesting’ looking drawings, uses a human fitness evaluator [24].

4.4.4 Selection

The ‘parents’ who are to provide the input to the next generation are selected based on their fitness: fit parents are selected preferentially over less fit ones.

Possibly the simplest process is to choose the top $n\%$ of the population. However, there is usually some random element in the selection process, to help preserve some diversity. With *roulette wheel selection*, the chance of being chosen as a parent is directly proportional to fitness value. This can sometimes lead to premature convergence if a badly sub-optimal but relatively very fit solution occurs early. This can be overcome with *ranked selection*, where the chance of being chosen is instead proportional to the parent’s fitness ranking.

These processes require the fitness of the entire population to be known. In some cases, this can be too expensive to calculate. *Tournament selection* overcomes this problem. Candidates are selected at random for a tournament, and the fittest of these goes on to become a parent.

Many algorithms allow the possibility of *elitism*: keeping the best of the previous generation in the next, to ensure good solutions are not lost because of failure to be selected, or unfortunate variation.

4.4.5 Inheritance and variation

Offspring chromosomes are derived from parent chromosomes by inheritance and variation. Inheritance is simple copying of the chromosome. Inherited material is varied by the *genetic operators* of ‘mutation’, and, in some EAs, of ‘crossover’, the combination of chromosomes from two parents.

Mutation is controlled by mutation probability parameters. The mutations possible depend on the data type of the alleles. For a binary bit string chromosome, each bit may be flipped with the parameterised probability. For real number alleles, the value may be changed probabilistically by a small random amount. For tree-shaped chromosomes, a mutation may involve randomly selecting a node, and replacing its subtree with a random subtree.

One-point crossover of strings involves selecting a random position in the strings, then taking the value of the first string up to this point, and the second string beyond. More complicated crossover arrangements, with multiple crossover points, are also used. The simplest versions of these schemes require all chromosomes to be the same length. It is also important to ensure a representation that remains valid after such an operation.

Crossover of tree-based representations is achieved by swapping subtrees. There is precious little biological inspiration to guide the design of GP crossover operators, because of the non-linear nature of the artificial chromosome being manipulated. However, GP still conforms to the original abstract concept of “inheritance + variation + selection = evolution”, despite its distance from the biological realisation of this concept.

4.4.6 Stopping condition

The stopping condition usually combines current best fitness and number of iterations.

The search stops if a good enough solution been produced. This requires setting some acceptable threshold fitness to be passed. The search also stops once a certain threshold number of generations been run. The result in either case is the current best member of the population.

4.4.7 Diversity and premature convergence

The whole aim of EAs is to provide a process that does not get trapped in poor local optima, but that has a good chance of finding the global optimum (or at least, a very good local one). Certain choice of the multitude of parameters governing the behaviour of any one algorithm can result in premature convergence to sub-optimal solutions, however, so these have to be chosen with care. This choice can be problem specific, and is currently more art than science.

Crossover is a mechanism that can move an offspring some distance from its parents in the search space, but once an allele value has disappeared from the population, crossover cannot reintroduce it. Mutation can, so is an essential diversity-maintaining mechanism.

Another way of increasing diversity is to inject some ‘new blood’ random individuals into the population. This needs to be done with care, since random individuals, especially late in a run, will usually be relatively unfit, and so eliminated almost immediately. The *clonal selection algorithm*, an artificial immune system algorithm with some interesting parallels to EAs, has automatic introduction of new individuals every generation [57].

4.5 Evolutionary algorithms in particular

4.5.1 Evolutionary Strategies and Evolutionary Programming: the early days

Some of the earliest work on EAs is known as “Evolutionary Strategies” (ES) [75] [2]. It is characterised by using real-valued chromosomes, directly representing solution space values of interest.

Originally this used only mutation, although more modern variants may incorporate crossover. The mutation rates are controlled by Gaussian probability distributions generated from *strategy parameters*. These parameters are not global and fixed. Rather, each chromosome can include its own value of these parameters, so that they also get mutated, in a form of self-adaptation. More advanced strategy parameters can be used to link mutation rates of different parameters.

ES uses deterministic selection of the fittest. They are characterised by two parameters. μ is the number of parents in the breeding population, and λ is the number of offspring generated. The constraint $\mu < \lambda$ ensures selection pressure: more offspring are generated than are allowed to breed in the next generation.

(μ, λ) strategies breed λ offspring from μ parents, then choose the next generation of parents to be the μ fittest of these offspring. $(\mu + \lambda)$ strategies breed λ offspring from μ parents, then choose the next generation of parents to be the μ fittest of the pool containing both the offspring and the current parents.

The original ES algorithm was (1+1)-ES with no self-adaptation. This means 1 parent is used to breed 1 child (by mutation), then the better of the 2 solutions is selected to become the next generation. This is equivalent to hill-climbing.

Another very early form of EA is known as “Evolutionary Programming” (EP) [33]. It is rather similar to ES, in that it also uses self-adapting mutation parameters, and only mutation. However, it uses tournament rather than deterministic selection. The original application was to evolve finite state automata to recognise and predict strings. The specially designed mutation operators over this space of finite state automata include changing the initial state, adding or deleting a state, adding or deleting or retargeting a transition, and changing a transition label.

This early work on EAs suffered from being somewhat ahead of its time: there was simply insufficient computing power to execute the algorithms except on relatively small problems. Now that computing power has increased so that the algorithms have become practical, interest in them has re-emerged.

4.5.2 Genetic Algorithms: incorporating crossover

Genetic Algorithms (GAs) were invented by John Holland [46] [47], but did not receive that much prominence until they were promoted by his student David Goldberg [37] [38]. Mitchell provides a good introduction to GAs [68].

GAs are the variant of EAs most closely based on biology (though still very far from its full richness and complexity), having linear chromosomes with mutation and crossover. The operation of GAs is well-analysed, and its performance characteristics explained in terms of the schema theorem and the related k -armed bandit theory [46], and the building block hypothesis [37]. (Curiously, however, the “compact GA”, a variant that represents an entire population as a probability distribution rather than a set of strings, also performs well [43], even though it cannot be using building blocks.) More recently, Nix & Vose have analysed GA performance using Markov chains [71].

4.5.3 Genetic Programming: complex phenotypes

Although there are earlier variants, the first major use of Genetic Programming (GP) was due to John Koza [49]. GP is a variant of GA where the chromosomes are computer programs. It is of particular relevance to evolving quantum programs, and the next section is dedicated to it.

4.6 Genetic Programming

Descriptions and applications of GP can be found in the series of books by Koza [49] [52] [53] [54], and also the collection by Kinnear [50]. Banzhaf *et al* provide a good introduction to GP [3].

4.6.1 Representation of programs

The program is usually represented as a tree structure, corresponding to an instance of the parse tree of the programming language. As mentioned earlier, the genetic operators that mutate and crossover trees can perform quite radical pruning and

grafting of entire subtrees. So it is important to use a programming language that can cope with such manipulations, that remains at least syntactically and preferably type-correct after such surgery. Lisp (for example [89]) is a favourite, for this reason. Also, purpose-designed domain-specific languages can be used. It can be necessary to constrain the genetic operators to produce correct trees, or to “fix-up” the trees after the operations.

4.6.2 Program as the means to generate the solution

In the simplest variant of GP, each chromosome is a program that is executed to generate a specific potential solution. For example, a chromosome might be a *turtle graphics* program to draw a specific circuit diagram, in a solution space of circuits; the resultant circuit is evaluated in terms of the objective function. These programs tend to be input-free programs that have a single behaviour, and so can be evaluated simply by executing them.

4.6.3 Program as the solution

In the more general case, the program being evolved is expected to work on a range of inputs. For example, a sorting program is expected to work on any permuted input, and a quantum algorithm is expected to work for an arbitrary number of qubits. In this case the program is required to work well for all its inputs, but clearly it is infeasible to evaluate its fitness on all its inputs. It is instead evaluated on a representative sample of inputs. (See section 4.7.1 on co-evolution for one way to determine this sample.)

4.6.4 Disruption and ADFs

Naïve use of GP can result in very little evolutionary progress, because of the way the GP genetic operators nearly always badly disrupt good solutions. It is rare for a child tree to be fitter than its parent under gross mutation or crossover. Care can be taken to devise operators that minimise disruption, by being sensitive to the context of the subtree, so helping to maintain structure.

A more important anti-disruption mechanism is Koza’s Automatically Defined Functions (ADFs) [49, chapters 20, 21] [52]. The idea is to encapsulate ‘good’ subtrees as functions, essentially adding new ‘useful’ alleles (terminals) that are automatically preserved against disruption. The program trees that undergo evolution comprise one result-producing branch that can include calls to ADFs, and several function-defining branches whose terminals include formal arguments.

Koza introduces further automatically defined structures, for similar reasons: iterations (ADIs), loops (ADLs), and recursions (ADRs) [53, chapters 6–8]. For quantum algorithms, ADLs in particular could be important, because some quantum circuit descriptions have a natural “loop” structure (for example, Shor’s algorithm).

4.6.5 Bloat

A famous problem of GP is that of *bloat*: some genetic operators allow trees to grow very quickly unless measures are taken to prevent this. This is perceived to be a problem, because bloat is nearly always ‘junk code’ that has no effect on the semantics of the programs. For example, it may be ‘unreachable’ code, such as a large *then* branch guarded by a *true* condition, or a large loop body with zero iterations. It has been suggested this bloat is analogous to *junk DNA* (biological introns).

Various anti-bloat measures are used, such as setting hard size and depth limits, having fitness function that prefers smaller programs, choosing operators that do not cause bloat. These are effective at stopping bloat, but may negatively impact GP performance by restricting intermediate solutions.

Most work on bloat concentrates on the (lack of) semantics of the bloated code. However, recent work by Daida suggests that there may be general constraints on the very shape of trees that can be naturally evolved using GP [19] [20] [21]. Certain shapes of tree appear to be very hard to achieve, independent of any functionality: it appears that GP has a strong structural bias to evolving inherently sparse trees. This has implications for the design of GP languages, and the class of solutions that can be found.

4.6.6 Typed languages: Grammatical Evolution, Enzyme GP

As mentioned earlier, the programming language manipulated by the GP genetic operators must be robust to their somewhat drastic surgery. Consequently, this has led much GP work to be done in untyped languages.

Approaches to incorporating types include using only highly constrained genetic operators to keep the trees type-consistent (which can result in premature convergence because there is too little freedom). Others include evolving type-free programs and add the typing information at a later interpretation stage. One such approach is Ryan & O'Neill's *Grammatical Evolution*, which uses a variable length linear genome to encode references to grammar production rules, and uses these rules to generate the program during the genotype to phenotype mapping, at which point type information can be added [72] [79]. Lones' *Enzyme GP* [62] [61] is another approach that can be used to solve the type problem. Rather than evolving whole trees, it evolves fragments of trees, which then 'assemble' themselves into full program trees. This assembly process can be adapted to observe the type constraints.

4.7 Variations on a theme

4.7.1 Co-evolution of test cases and programs

Biological organisms do not evolve in some static unchanging fitness landscape: they co-evolve in dynamic along with other species, and their own landscape is changed in response to changes in these other species.

EAs can also use concepts of co-evolution: as a solution gradually improves, the problem can be made gradually harder by co-evolving it. This can help the problem of premature convergence, where an initially random population is almost uniformly useless at solving the full problem, and survival is dominated by a few 'lucky' individuals who are nevertheless not near a global optimum. If the problem starts off in an easier form, then a wider range of initial population might be able to 'survive' it.

Similarly co-evolution can be used to help with the selection of GP evaluation inputs. A program's fitness should ideally be evaluated against all its inputs, but this is infeasible in general, and a sample of inputs is used. But which sample? One approach is to evolve the program to perform well on its input sample, whilst simultaneously co-evolving the *sample* so that the program has *poor* performance [44]. Thus the input sample evolves to exercise the program to its utmost.

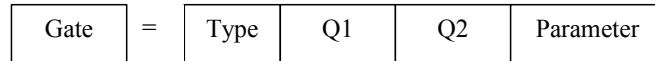


Figure 9. Gate template.

4.7.2 Evolving the evolutionary parameters

As noted earlier, there can be many parameters in EAs, such as population size, tournament parameters, choice of mutation and crossover operators, mutation rates. Much (often ineffectual) effort goes into ‘tweaking’ these to get the ‘best’ results.

Another approach is to follow the route of Evolutionary Strategies, and include any and all of these parameters as strategy parameters in the chromosome so they evolve along with the solution. This *self-adaptation* reduces the number of parameters, but at the expense of making the search space larger.

4.7.3 Intrinsic evolution

Some of the most exciting new work in the area of EAs is on *intrinsic evolution*, where the phenotype is not a computer representation of the solution space, but the physical solution space itself. For example, evolutionary hardware evaluates the actual hardware circuits, not software simulations of those circuits [99]. This can lead to novel discoveries, as the embodied solution exploits physical characteristics abstracted away from in a simulation [92].

5 Evolving Quantum Algorithms: implementation issues

In this section we review some implementation issues that arise when evolutionary algorithms are used to search for quantum algorithms.

5.1 Representation of Potential Solutions

5.1.1 Direct Encodings

A circuit can be represented as an ordered series of gates. The order in which gates appear in the list is the order in which the corresponding transformations are to be applied. A gate template is simply a sequence of slots, with each slot being instantiated to attribute values. In the template shown in Figure 9 there is a slot for the type of gate (I, X, Y, Z, N, CN, U , etc.), two slots for the identifiers of the qubits upon which the gate operates, and a slot for a (further) parameter. All gates have a type and at least one operational qubit. The remaining slots are interpreted conveniently for each gate type, or ignored where appropriate.

A NOT gate acting on qubit 3 is represented as the quadruple $(N, 3, *, *)$, where ‘*’ means we do not care what values are in the slot: they are ignored. A controlled-NOT gate with control qubit 3 and target qubit 1 is represented by $(CN, 3, 1, *)$. A single qubit rotation $U(\pi)$ on qubit 5 is represented by $(U, 5, *, 3.14159)$, and so on.

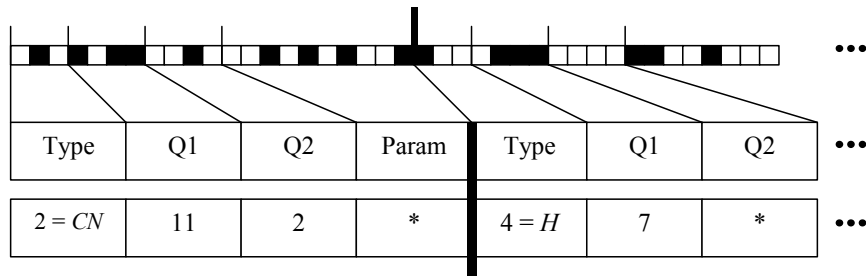


Figure 10. Direct encoding

In practice, there are choices as to how the series of gates is represented at a low level. The most basic representation is as a bit string. (Bit string representations are still very common in genetic algorithm applications.) Figure 10 illustrates how subsequences of bits might map onto gate fields. With black squares denoting bit values of 1 and white squares denoting bit values of 0, we can see how a string of bits (e.g. a bit chromosome) can be decoded as a circuit.

Such encodings are not without their problems. Suppose there are 5 gate types. This requires at least three bits to represent. We may ensure that an initial population has type fields with bit values of 000, 001, 010, 011, 100 (that is, between 0 and 4), but simple crossover operations are likely to produce 101, 110, and 111. These will need to be interpreted in some way. Interpreting the value modulo 5 produces an acceptable type index, e.g. 110 denotes 6, and $6 \bmod 5$ is equal to 1, and so 110 would represent gate type 1. 0 would be represented by (000,101), 1 by (001,110), 2 by (010,111) but 3 and 4 would have the single representation 011 and 100 respectively. Thus, some elements of the space are over-represented, possibly biasing certain types of search. The most common bit string mutation operator is the simple bit flip. The resulting field values will need interpreting in the same way.

Some researchers have used similar simple bit string representations with genetic algorithms. Although such representations are considered unsophisticated by the evolutionary computation community, their application is not without some success (see the gate implementation work described in section 6.3, and the teleportation circuits of Yabuki and Iba described in section 6.5).⁷ It is of course possible to work directly on character, integer and real fields.

5.1.2 Linear List Encodings

Whereas in traditional tree-based genetic programming programs are expressions in a functional language such as LISP (see below) *linear genetic programming* typically uses varying length *lists* of imperative programming language instructions [3]. There is generally no *a priori* reason for expecting a specific length of solution, and this approach allows simple manipulation of populations with individuals of varying lengths. Substituting classical instructions for quantum gates provides us with a natural representation and powerful approach for the evolution of quantum circuitry. Williams & Gray's GP approach [97] exemplifies the flexibility afforded by such schemes, with a variety of evolution operators provided: mutation, substitution, crossover, transposition, insertion and deletion. Figure 11 illustrates a generalised crossover operation.

⁷ This success may be in spite of the representation used.

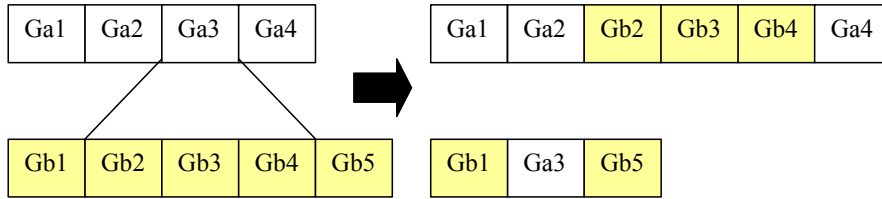


Figure 11. Flexible Crossover with List Representations

Linear GP with classical program evolution also allows single operations to be skipped over via preceding branching instructions. (Williams & Gray do not incorporate such features).

Spector *et al* [86] briefly describe two linear genetic programming variants: stack based linear genome genetic programming (SBLGP) and stackless linear genome genetic programming (SLLGP). SBLGP represents programs as linear lists of functions that communicate via a global stack (thus the approach generalises away from quantum gate lists to instruction/function lists). SBLGP lends itself better than functional programming tree based approaches to the evolution of programs whose working functionality is implemented by side effects (see below). The SBLGP also allows for certain structuring mechanisms to be incorporated. Spector *et al* report that SBLGP was generally favoured over the traditional tree-based approaches described below. Stackless GP uses a linear list of gates much as described above for Williams & Gray [97]. Spector *et al* point out that this may be entirely appropriate when scalability is not an issue (and so the structuring mechanisms such as parameterised iteration are unnecessary).

Linear genetic programming appears to be a powerful and flexible approach to evolutionary computation. The approach seems naturally suited to quantum program evolution since quantum programs are inherently sequential, and the implementation seems simpler than for traditional tree based approaches.

5.1.3 Spector *et al*'s Traditional GP Tree Encoding

With some linear list variants the representations code for specific solutions to specific problems. The solution space may be, for example, the set of circuits operating over 4 qubits. An evolved solution might work perfectly over 4 qubits but simply be inapplicable to a similar problem with 5 qubits.

There is a need to derive *scalable* artefacts and human understanding of evolved artefacts may be an important goal for evolutionary search in the quantum domain. A feature of scaleable human-developed artefacts is the use of *structure*, because structure captures an intellectual and communicable idea. As a simple example, an adder circuit comprises a connected series of single bit adders. In classical computing, circuitry for a 12-bit adder looks a lot like that for a 10-bit adder. Both are generally built using the same overall approach, the difference being that for the 12-bit case the underlying structural idea is repeated twice more. Modern programming languages have significant structuring mechanisms: if-then-else; for-loops, while-loops; functions; procedures etc. Functions and procedure provide high-level reusable building blocks. Furthermore these are often parameterisable (for example, an integer array sort routine will generally accept arrays of different lengths). We would like similar facilities to be provided for the evolution of quantum algorithms.

This is addressed by using *second order encodings*. With direct encodings we evolve a circuit directly in one step. With second-order encodings, we evolve a

program that when executed produces a circuit. This circuit-generating program can be run with various parameters to generate different circuits. For example, if the program is parameterisable in the number of qubits we could use it to generate circuits for 3, 4, and 5-qubits problems, and so on. We can see that the ability to incorporate structuring mechanisms such as parameterised iteration is important.

The early GP work by Spector *et al* (see section 6.1.1) uses such an approach. Functions parameterised by numbers add gates to the current circuit; the initial circuit is empty. Subprograms (subtrees) return numeric values that are either used directly or after coercion to integers as parameters of the parent node. The closure type is 'number'; this includes integers, rationals, floating point numbers, and complex numbers.

The approach has various functions to add some standard gates to the circuit. Examples are:

- H-GATE: adds a Hadamard gate to the end of the circuit. It has one parameter that is coerced to a valid qubit index. It returns its argument as a result.
- U-THETA-GATE: adds a rotation gate to the circuit. There are two parameters: the first is coerced to be a valid qubit index, the second is an angle in radians.
- CNOT-GATE: adds a controlled-NOT gate at the end of the circuit. It has two parameters, coerced to form valid source and target qubits. The first argument is returned as a result.

Iteration constructs are incorporated in the program, such as:

- ITERATE. This takes two parameters: the second is some subprogram body; the first is coerced to a non-negative integer denoting how many times that body is to be executed.
- IQ. This has a program body as its single parameter. This body is executed a number of times equal to the number of qubits in the system.

A variety of helpful support functions are provided (e.g. mathematic operations such as +, -, *, etc.). The iteration constructs are very important. They allow the system to evolve *scalable* algorithms: algorithms that can be parameterised to be used on systems of different sizes.

The language has a general LISP flavour and representation. Consider the program.

```
( CNOT
  ( U-THETA
    PI
    (/ PI 2.0)
  )
  ( H-GATE (+ 1 (/ PI 2.0))
  )
)
```

When executed this will produce

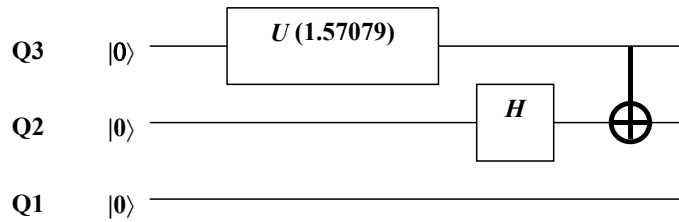


Figure 12. Circuit generated when program is executed.

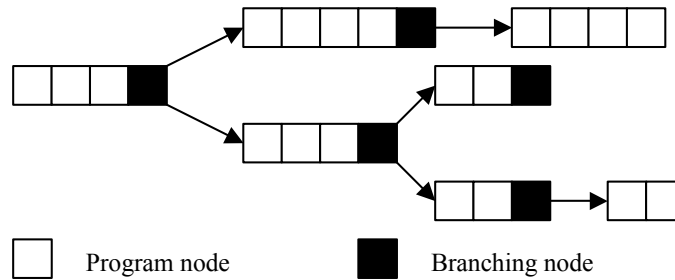


Figure 13. Linear Tree GP program representation.

U-theta	qubit: 3	theta: 1.57079	// PI = 3.14150 coerced to 3
Hadamard	qubit: 2		// 1 + PI/2 = 2.57079 coerced to 2
Controlled-not	control: 3	target: 2	// PI = 3.14150 coerced to 3 // 2.57079 coerced to 2

This program produces the circuit shown in Figure 12.

A typical problem with basic GP approaches is that they are weakly typed. The coercion of returned values to parameters is somewhat unconvincing and represents a significant potential restriction on the programs that can practically be evolved. Spector *et al* [86] also note that tree representation comes at a cost in terms of time, space and complexity, with “no guarantee that they are the most appropriate representation for all problems”. A major motivation behind the approach – the search for structure and scalability – is entirely well-founded.

5.1.4 Leier and Banzhaf’s Linear Tree GP Representations

Kantschik & Banzhaf [48] introduce a new tree-based representation for GP termed *linear tree GP* (LTGP). In LTGP a program comprises linear instructions sequences connected by branching instructions, Figure 13. A path from the root node to a leaf node defines an execution. The aim is largely to allow programs to execute different instructions sequences for different inputs. Leier & Banzhaf [56] have adapted this scheme for evolving quantum programs. Unitary transformations form the program instructions and measurements form the branching nodes (with the 0 and 1 branches being executed in the context of those measurements having occurred). Both branches may be executed if the branching probabilities are non-zero. The reader is referred to [56] for details.

5.2 Spector’s Push-Based System

The most advanced suite of quantum genetic programming tools so far is due to Spector. A good deal of his recent book [88] is given over to explaining the basics

of the underlying technological tools. PUSH is a Lisp-like programming language with very simple syntax:

$$program ::= instruction \mid literal \mid (program^*)$$

There are several stacks for different data type operations. Thus may be stacks for integers, Booleans, floats and so on as well as a code stack. New stacks can be added (e.g. a quantum gate stack). The system allows variable names to be associated with elements (including code fragments) and has features to ensure safe operation (such as ignoring instructions when there are insufficient arguments on the appropriate stack).

Execution of a program (P) is a recursive application of:

If P is a single instruction then execute it.
 Else if P is a literal push it onto the appropriate stack.
 Else P is a list: sequentially execute each of the programs in the list.

The first component of the program

$$((5\ 4\ \text{INTEGER.}+) (2.0\ 2.0\ \text{FLOAT.*}))$$

causes 5 and then 4 to be placed on the integer stack, the 4 and 5 to be popped from the integer stack and added, with the result (9) being placed back on the integer stack. Similarly the effect of the second component's execution is to place 4.0 on the float stack. The reader is referred to [88] for further details.

PUSH GP is a genetic programming system that evolves programs in the PUSH language. The system allows multiple data types, modularity features, support for recursion and support for code-self development. It supports some fairly traditional GP operator features.

Spector provides detailed results of applying this system to solution of various problems: Scaling Majority On, Deutsch-Josza XOR, OR and AND-OR, Grover's search (4-item database) and some gate communications problems. The facilities described in [88] (including visualisation of simulations) collectively form a cohesive quantum genetic programming research suite. The underlying simplicity of the supporting technology is striking.

5.3 Evaluating Candidate Solutions: Cost and Fitness Functions

Evaluation functions define what it means to be a desirable solution to a problem and provide *guidance* to the search process to reach such a goal. Williams & Gray [97] note: "We regard the most sensible evaluation measure as an open question". This remains the case at the time of writing this review (2004).

A variety of evaluation functions have been used. We identify and examine three broad types.

5.3.1 Evaluation Based on Deviation from Target Matrix

In their approach Williams & Gray [97] assume that there is a target unitary matrix U and the task is to evolve a circuit with unitary matrix S that implements it. They aim to perform what computer scientists would term *refinement*: breaking a higher level construct down into the composition of more concrete (lower-level) ones. Their cost function is given by

$$f(S, U) = \sum_{i=1}^{2^n} \sum_{j=1}^{2^n} |U_{ij} - S_{ij}|^R$$

This is an intuitively appealing function and it is applied with some success (see below). The choice of magnitude of differences is not definitive. Although William & Gray set the value of R to 1, DiVincenzo & Smolin [30] use $R = 2$ to ‘punish’ deviation from a target matrix. Non-integral values might prove useful: Clark *et al* [18] demonstrate the sensitivity of some problems to exponent choice.

Lukac *et al* [63] present a detailed account of the evolution of circuitry (principally lower level implementations of important ‘gates’) with various evaluation functions that combine functional correctness (with an error component based on matrix element deviations as above) and circuit cost.

5.3.2 Evaluation Based on Deviation from Target Amplitude Vectors

We might not know the unitary transformation we desire but we may be able to indicate its likely desired effect on some test inputs, that is, we may be able to define properties of a desired final amplitude state vector and punish deviation from them.

Yabuki & Iba [98] use three test cases (fitness cases) for evaluating the fitness of their evolved teleportation circuits. This is based on the degree of similarity of the received qubit value with the source qubit to be teleported. At first sight, it may seem unusual for so few test cases to be needed. On reflection, the reader might find it difficult to conceive of a circuit that successfully teleports three random qubit states that is *not* a generally applicable teleportation circuit.

For the evolution of deterministic circuits Massey *et al* [65] use a cost function given by

$$f = \sum_i (\| V_{Ti} - V_{Ri} \|)$$

where V_{Ti} is the target amplitude vector for the i th input test case, and V_{Ri} is the amplitude vector achieved after applying a candidate circuit to the i th input. A further nuance can be seen when we wish to evolve circuits that ‘bump up’ the magnitude of the amplitudes of results we wish to see. Here the exact amplitudes of the resulting state vectors may not be crucial. Rather, it is the probability that matters, and so we can base cost functions on the deviation in *magnitude*.⁸

Spector *et al*'s work [4] [84] [85] [86] has a probabilistic notion of success and uses a fitness function that captures functional correctness but also aspects of efficiency. It has the form:

$$f = \text{hits} + \text{correctness} + \text{efficiency}$$

The *hits* component is the total number of fitness cases used minus the number of fitness cases where the program produces the correct answer with a probability of more than 0.52 (chosen to be far enough away from 0.5 to be sure it is not due to rounding errors). The *correctness* component is defined as:

⁸ Whether this matters or not depends on what you are evolving the circuit for. If you want to observe a ‘result’ then it is largely the probabilities that matter (and so issues of phase etc. are of no concern); if you want to use the circuit as a component in a wider circuit then amplitude is generally important.

$$correctness = \frac{\sum_{i=1}^n \max(0, error_i - 0.48)}{\max(hits, 1)}$$

Because it is desirable for the fitness function to focus on attaining probabilistically correct answers to all fitness cases, rather than simply improving the probability of success in those fitness cases where it is already good enough (e.g. from a 55% success rate to a 60% success rate), errors smaller than 0.48 are ignored. Also, it is desirable that reasonably fit programs are compared primarily with respect to the number of fitness cases they produce a (probabilistically) correct answer for, and only secondarily with respect to the magnitudes of the errors of the incorrect cases, the ‘pure’ *correctness* term is divided by *hits* (unless *hits* < 1) before being used in the fitness function.

The *efficiency* is the number of quantum gates in the final solution, divided by a large constant. Therefore, efficiency has a very small effect on the overall fitness of the solution, until programs are evolved that solve all fitness cases, at which point the other two terms become zero and the efficiency dominates. The overall effect is that the search initially concentrates on finding probabilistic solutions to the problem, and then tries to make those solutions more efficient, in terms of the number of quantum gates used. No effort is wasted on trying to make the solutions more accurate (*i.e.* increase the probability of them correctly giving the answer).

The fitness function of Spector *et al* has been adopted by Massey *et al* [65] for probabilistic circuits. More general fitness functions (but using many of the same concepts) can be found in Spector’s book [88]. Spector *et al* [86] note that the fitness function evolved as the work reported progressed. Further fitness function details can be found in [84].

5.3.3 Evaluation Based on Resource Usage

Not all searches start from nothing. If you have a working circuit you may wish simply to improve it in some way. Compilers for traditional programming languages generally have an optimisation engine that applies a series of functionality preserving transformations to obtain a program that improves some non-functional aspect such as average execution speed.

Similar considerations apply to quantum circuitry. Work has been carried out to determine circuit identities (for example, [60]). Maslov *et al* [64] discuss *linear cost circuit metrics* and *non-linear circuit cost metrics* (the former being a simple weighted gate count, the latter being based on the full circuit).

Concentrating solely on efficiency (however defined) simplifies matters: functionality and efficiency may often be in conflict and fitness and cost functions may be inclined to produce tradeoffs we would not want. It remains, however, an open question whether it is best to evolve a circuit then optimise it, or else evolve an efficient circuit in one go.

5.4 Structure of the search landscape

The structure of the search landscape has a strong effect on the ease of searching it. Rugged landscapes are difficult to search, because the fitness of the current position gives little indication of the fitness of nearby positions. Landscapes with many local optima can “trap” the search. Some of these problems may be alleviated by choosing the landscape with care. There are three factors under the control of the

designer: (1) the points in the search landscape itself, determined by how the problem is represented; (2) the “height” of each point, determined by the fitness function; (3) the move function, or which points can be reached from which other points, determined by the choice of genetic operator.

A principled design of the search space needs understanding of how these various choices affect it. Leier & Banzhaf [57] investigate the shape of the search landscape for a particular case of the 2- and 3-qubit Deutsch-Josza problem (with a predetermined gate set and fitness function), for a range of program sizes (10 to 30 gates), and for mutation operators only. They investigate ruggedness by estimating the *autocorrelation function* of time series generated by random walks around the search space, where the paths are given by the mutation operators. They investigate the structure of local isolated optima by estimating certain *information measures*.

Their results of low autocorrelation indicate extremely rugged landscapes: “beyond 2 steps most of the points on the landscape path become almost uncorrelated”. The autocorrelation is slightly larger for $n = 3$ qubits than for $n = 2$, and also for larger programs. The information measure also shows larger program sizes tend to have smoother landscapes, but also have a more complex structure of local optima.

It is difficult to interpret what the combined effect of these opposing trends in ruggedness and local optima might be for even larger program sizes or higher number of qubits, and whether any improvements in searchability are outweighed by the exponentially increasing size of the search spaces. However, Leier & Banzhaf [57] provide an important first investigation, possibly demonstrating why search for (small) quantum programs is proving quite tricky. Further investigation of landscape structure in terms of larger programs, more sophisticated genetic operators, and different fitness functions, is called for.

5.5 Hand Processing

Sometimes the mechanisms by which the search proceeds give rise to circuits that can be simplified. For example, two successive applications of the Hadamard operation to a qubit (without any intervening operation in the system) produces no effect ($H^2 = I$) and so such a pair of H gates can be removed. Such ‘junk’ may actually serve a purpose during an evolutionary search, but at the end it is simply clutter. Various authors have resorted to hand simplification. Such removals are particular examples of the more general idea of semantics-preserving operations. Traditional program compilers carry out a variety of such operations to produce more efficient code. These are taken from a set of substitution templates derived over many years of experience. The authors believe that a similar quantum circuit substitution library can be created, and highly efficient circuits created from inefficient, but functionally correct, ones. This issue is discussed in section 6.4.

5.6 Simulation Issues

Evaluation of the cost function requires simulation of a quantum computation on a classical hardware platform. Simulation efficiency is of major practical importance. Spector acknowledges such issues in his book [88]. Consider the issue of how a unitary transform should be stored. The simplest would be to store its matrix, but this will become unmanageable as the number of qubits in a system grows (it requires 2^{2n} elements to be stored). A 15-qubit system would require over a billion entries to be stored per matrix. As indicated by [88], for some operations it suffices

to use an operation that has the same effect on the state amplitude vector. (Spector refers to this as “implicit matrix expansion”.) For example, it is fairly pointless to store a fully lifted NOT operation. It is simpler to invoke a program that effectively swaps corresponding pairs of amplitudes: if $|0x\rangle$ has amplitude p_{0x} and $|1x\rangle$ has amplitude p_{1x} then the NOT operation on the first qubit simply swaps the two amplitudes. Massey *et al* [65] refer to this particular optimisation as an example of *row swapping*. Spector’s implicit matrix expansion is more general. Spector provides algorithms for explicit matrix expansion (what we have termed ‘lifting’) and for applying implicitly expanded gates. Explicit matrix expansion of a gate may be necessary, e.g. for use in forming some explicit product matrix.

6 Evolving Quantum Algorithms: results

In this section we review how evolutionary algorithms have been used to discover quantum algorithms.

6.1 Circuits for Classic Combinatorial Problems

6.1.1 Spector *et al*: Deutsch-Josza Promises, Grover’s Search, ORs and AND-ORs

A number of papers authored in various combinations by Spector, Bernstein, Barnum & Swami established the field of quantum genetic programming [4] [5] [84] [85] [86].

Some of the earliest work using GP aimed to evolve quantum circuitry to determine properties on oracle functions: given a quantum black-box function $f(q_1, \dots, q_n)$ determine whether it has the property $P(f)$. We have already seen the (non-evolved) Deutsch-Josza algorithm to determine whether a function f is balanced or constant (for a 1-input function f this is the parity problem). [86] presents an evolved solution to the corresponding 2-bit promise problem (using traditional tree-based GP).

[86] also describes the evolution using SBLGP (see section 5.1.2) of an instantiation of Grover’s algorithm for solving the four-item database problem. (The database is defined by an oracle function $f(x)$ over $0..3$ and the aim is to return the single index in that range for which $f(x) = 1$, i.e. there is a single ‘marked’ solution.) This is important because Grover had published his algorithm as recently as in 1997. Other circuits can be found in [88]. It is not uncommon for human analysts to simplify evolved artefacts. (This is indeed a very good thing to do.) Figure 14 shows a hand-simplified version of Grover’s solution to the four-item database.

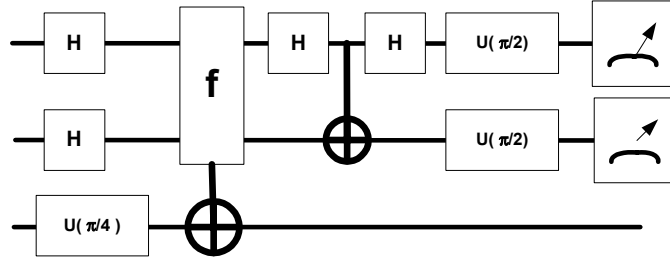


Figure 14. GP evolved (but hand simplified⁹) Grover's four-item database search.

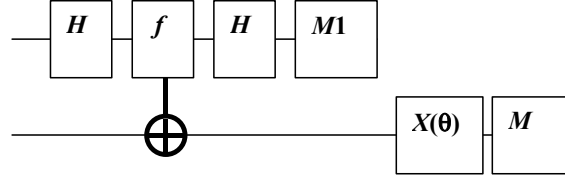


Figure 15. Circuit addressing the OR problem.

Two other simple fundamental properties concern ORs and ANDs of ORs. The OR problem is simple: determine whether any input x gives rise to a true output $f(x)$. For a Boolean function $f(x)$ on n variables the AND-OR problem considers a complete balanced binary tree with leaves labelled left to right with the function values $f(0), f(1), \dots, f(2^n - 1)$. The AND-OR function interprets this tree as a Boolean expression tree with root AND node and nodes alternating between OR and AND as paths are traversed from root to leaves. For 1, 2 and 3 inputs the AND-OR(f) formulae are:

$$AND/OR_1(f) = f(0) \wedge f(1)$$

$$AND/OR_2(f) = (f(0) \vee f(1)) \wedge (f(2) \vee f(3))$$

$$AND/OR_3(f) = ((f(0) \wedge f(1)) \vee (f(2) \wedge f(3))) \wedge ((f(4) \wedge f(5)) \vee (f(6) \wedge f(7)))$$

Barnum *et al* [5] used SLLGP to evolve a faster than classical solution to the OR problem. For the one qubit case an evolved circuit is shown in Figure 15.

With initial state $|00\rangle$, application of the first three gates produces the state

$$\frac{1}{2}(|0\rangle(|f(0)\rangle + |f(1)\rangle) + |1\rangle(|f(0)\rangle - |f(1)\rangle))$$

The measurement gate $M1$ terminates the computation if a 1 is measured and the computation continues otherwise. The result of the evaluation of $f(0) \vee f(1)$ is taken to be $M1$ if it returns a 1, or else the result is taken to be $M2$. The $X(\theta)$ is defined by

$$X(\theta) = \begin{pmatrix} \cos \theta & \sin \theta \\ \sin \theta & -\cos \theta \end{pmatrix}$$

⁹ Spector [88] reports that this circuit is a simplification (by Bernstein) of a GP-evolved circuit.

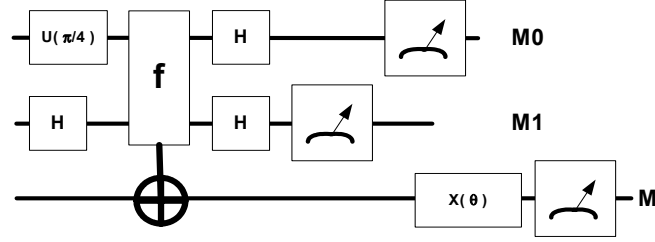


Figure 16. Faster than classical solution to 2 Bit AND-OR, theta=0.74909.

f(00)f(01)f(10)f(11)	p _e	f(00)f(01)f(10)f(11)	p _e
0000	0.00560	0101,0110,1001,1010	0.28731
0001, 0010, 0100,1000	0.28731	1101, 1110, 1011, 0111	0.21269
0011,1100	0.21269	1111	0.00560

Figure 17. Error probabilities for 2-bit AND-OR solution.

Let the four possible functions be f_{00}, f_{01}, f_{10} , and f_{11} . For f_{00} and f_{11} there is zero probability of observing a 1 on the first (upper) qubit. For f_{01} and f_{10} there is a probability of $\frac{1}{2}$ of (correctly) observing a 1. If a 0 is measured then the follow states result:

$$|00\rangle \text{ for } f_{00}; |01\rangle \text{ for } f_{11}; \frac{1}{\sqrt{2}}|0\rangle(|0\rangle+|1\rangle) \text{ for } f_{01} \text{ and } f_{10}$$

After applying $I(0)$ the following states are achieved

$$|00\rangle \text{ for } f_{00}; -|01\rangle \text{ for } f_{11}; \frac{1}{\sqrt{2}}|0\rangle(|0\rangle-|1\rangle) \text{ for } f_{01} \text{ and } f_{10}$$

This is an important theoretical result in its own right. Previously one-bit XOR had been shown to be amenable to faster than classical quantum solution. One-bit OR had now been shown similarly improved by quantum computational means.

A faster than classical solutions to the two bit AND-OR problem has also been evolved using SLLGP. Again, some hand-tuning was used to improve the evolved algorithm. The circuit diagram is shown in Figure 16 with error probabilities p_e for the various functions f shown in Figure 17.

6.1.2 Leier & Banzhaf: Evolution of Hogg's Algorithm

Hogg [45] has demonstrated efficient quantum algorithms for attacking k -sat problems. Let V_1, \dots, V_n be Boolean literals, and let L_i be the literal V_i or its negation. Given a formula that is the conjunction of disjunctions of k L_i

$$(L_{11} \vee L_{12} \vee \dots \vee L_{1k}) \wedge (L_{21} \vee L_{22} \vee \dots \vee L_{2k}) \wedge \dots \wedge (L_{m1} \vee L_{m2} \vee \dots \vee L_{mk})$$

find an assignment for the V_1, \dots, V_n that satisfies the formula. A simple 2-sat formula and an assignment that satisfies it is:

$$(V_1 \vee \neg V_2) \wedge (\neg V_1 \vee V_2); \quad V_1 = \text{true}, \quad V_2 = \text{true}$$

		H 0
	H 0	H 1
H 0	H 1	H 2
H 1	H 2	H 3
INP	INP	INP
Rx[3/4 Pi] 0	Rx[3/4 Pi] 0	Rx[3/4 Pi] 0
Rx[3/4 Pi] 1	Rx[3/4 Pi] 1	Rx[3/4 Pi] 1
	Rx[3/4 Pi] 2	Rx[3/4 Pi] 2
		Rx[3/4 Pi] 3

Figure 18. Solutions to 1-Sat on 2, 3 and 4 variables ($Rx[\theta]$ is a rotation).

Leier & Banzhaf [56] have evolved circuits equivalent to Hogg’s algorithm for the simple 1-sat case. Though classical algorithms for this problem are of $O(n)$, Hogg’s algorithm is still more efficient. Interestingly, they present some “slightly hand tuned quantum algorithms” arising from GP searches for 1-sat on 2, 3 and 4 variables, given in Figure 18.

We can readily see there is a pattern suggesting extension of the idea. Indeed, Leier & Banzhaf refer to “evidently and ‘visibly’ scalable algorithms, which correspond to Hogg’s algorithm”. This is useful since evolving quantum algorithms is likely to be tricky, as they note:

The problems of evolving novel quantum algorithms are evident. Quantum algorithms can be simulated in acceptable time only for very few qubits without excessive computer power. Moreover, the number of evaluations per individual to calculate its fitness are given by the number of fitness-cases usually increases exponentially or even super-exponentially. As a direct consequence, automatic quantum circuit design seems to be feasible only for problems with sufficiently small instances (in the number of required qubits). Thus the examination of scalability becomes a very important topic and has to be considered with special emphasis in the future. — [56]

Using GP (or other search techniques) to evolve small circuits that can be analysed by researchers seems a promising way forward. Search needs only to *augment* human ability, it doesn’t need to solve every problem we throw at it.

6.2 Deterministic to Probabilistic: Massey *et al.*

Massey *et al* [65] report the results of using two quantum genetic programming suites: QPACE-II and QPACE-III. QPACE-II uses a direct encoding, whilst QPACE-III uses a second order encoding (where the program is executed to generate a circuit).

Q-PACE II evolved a deterministic full adder circuit using simple and controlled versions of the N and H gates, and a non-unitary zeroing gate Z . Q-PACE II found the solution previously designed by Gosset [39]. The authors report that the evolution of quantum arithmetic circuitry seems very hard, with more challenging problems remaining unsolved by the approach, even after a multi-stage approach was adopted. So they moved away from the search for deterministic circuits to a search for probabilistic circuits. Q-PACE II found a probabilistic half-adder on 3 qubits using only the H gate and the zeroing gate Z (together with their controlled equivalents). The problem is defined as $|x,y,z\rangle \rightarrow |x, x \text{ XOR } y, x \text{ AND } y\rangle$, where $|x \text{ XOR } y\rangle$ is the sum bit and $|x \text{ AND } y\rangle$ the carry bit. Q-PACE II evolved the circuit shown in Figure 19, with results shown in Figure 20.

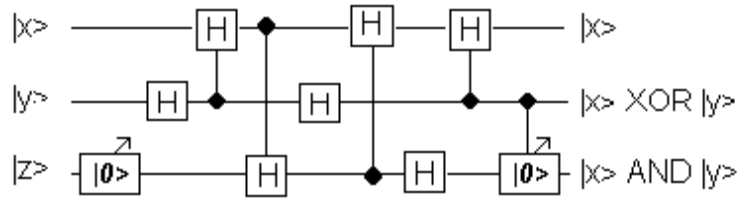


Figure 19. Probabilistic half-adder.

Initial State		Correct Answer	Probability of ending in each state, after running the circuit							
			000>	001>	010>	011>	100>	101>	110>	111>
000>	→	000>	0.53	0.22	0	0	0	0.09	0.14	0
001>	→	000>	0.53	0.22	0	0	0	0.09	0.14	0
010>	→	010>	0	0.05	0.61	0	0	0.09	0.24	0
011>	→	010>	0	0.05	0.61	0	0	0.09	0.24	0
100>	→	110>	0.09	0.04	0.03	0	0	0.02	0.82	0
101>	→	110>	0.09	0.04	0.03	0	0	0.02	0.82	0
110>	→	101>	0	0.30	0.10	0	0.02	0.53	0.04	0
111>	→	101>	0	0.30	0.10	0	0.02	0.53	0.04	0

Figure 20. Probabilities of obtaining outcomes for half adder inputs.

Fitness Case	Correct Answer	Probability of ending in each state, after running PF MAX 1			
		00>	01>	10>	11>
(3, 1, 0, 2)	00> (i.e. $x = 0$)	0.53	0.22	0.03	0.22
(0, 2, 3, 1)	10> (i.e. $x = 2$)	0.03	0.22	0.53	0.22
(3, 0, 1, 2)	00> (i.e. $x = 0$)	0.53	0.22	0.03	0.22
(1, 2, 3, 0)	10> (i.e. $x = 2$)	0.03	0.22	0.53	0.22
(3, 2, 0, 1)	00> (i.e. $x = 0$)	0.53	0.22	0.03	0.22
(2, 3, 0, 1)	01> (i.e. $x = 1$)	0.22	0.53	0.22	0.03
(2, 0, 1, 3)	11> (i.e. $x = 3$)	0.22	0.03	0.22	0.53
(2, 1, 3, 0)	10> (i.e. $x = 2$)	0.03	0.22	0.56	0.19

Figure 21. Results for Permutation Function Fitness Cases.

Q-PACE III evolved a number of probabilistic quantum programs which, when given a number of suitably encoded $[0..3] \rightarrow [0..3]$ permutation functions, returned for every one of these permutation functions (with a probability > 0.5) the value of x that gave the maximum value of $f(x)$ for that function. (This is called the “PF MAX” problem for short.) Ultimately, Q-PACE III evolved a program that ‘solved’ the problem for all 24 possible $[0..3] \rightarrow [0..3]$ permutation functions, as shown below.

Q-PACE III evolved the program “PF MAX 1” using the following 8 fitness cases (expressed as permutations): $\{(3,1,0,2), (0,2,3,1), (3,0,1,2), (1,2,3,0), (3,2,0,1), (2,3,0,1), (2,0,1,3), (2,1,3,0)\}$. The result of PF MAX 1 is shown in Figure 21.

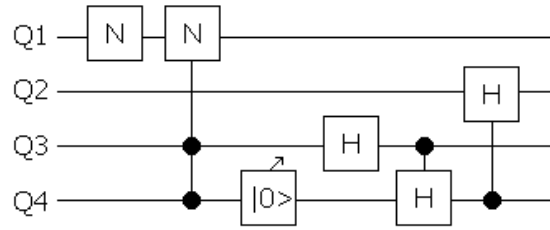


Figure 22. Probabilistic PF-MAX circuit.

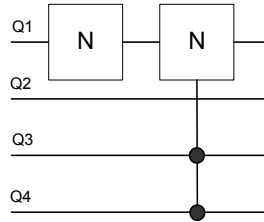


Figure 23. Second PF-MAX circuit.

PF MAX 1 is a probabilistic solution to all 8 of the fitness cases used, and for 20 out of the 24 possible permutation functions, it gives the correct answer with a probability of more than 0.5; for the other 4 fitness cases, it gives the correct answer with a higher probability than any given incorrect answer. Thus PF MAX 1 seems a true MAX algorithm for $[0..3] \rightarrow [0..3]$ permutation functions, that “works” on all 24 of these functions. Although evolved from only 8 fitness cases, the resulting PF MAX 1 is much more general. The circuit (after hand removal of 5 gates that have no effect) is shown in Figure 22.

Repeated experiments failed to evolve a program that would give the correct solution with $p > 0.5$ for *all* 24 fitness cases. Relaxing the acceptance criterion to $p > 0.4$ enabled Q-PACE III to evolve a single quantum circuit with $p = 0.5$ of returning the correct answer for *all* the 24 fitness cases (the probabilities of returning incorrect answers are 0.25 or zero). So the quantum circuit implements a probabilistic MAX function that has twice the probability of “guessing”. The circuit generated (after hand removal of several gates that have no effect) is shown in Figure 23.

The system is exploiting the initial set-up very efficiently. Suppose, for example, that the maximum occurs at $x=00$. Then $|0011\rangle$ has amplitude $\frac{1}{2}$ (corresponding to probability $\frac{1}{4}$) and $|0000\rangle$, $|0001\rangle$ and $|0010\rangle$ all have amplitude of 0. Now consider $x=10$. We must have $f(10)=00$, $f(10)=01$, or $f(10)=10$ since the maximum is already reached uniquely by $f(00)=11$. Suppose $f(10)=00$. Then the state $|1000\rangle$ has amplitude $\frac{1}{2}$, while $|1001\rangle$, $|1010\rangle$ and $|1011\rangle$ all have amplitudes of 0. The application of the CCN operation transforms $|1000\rangle$ to $|0000\rangle$ with amplitude $\frac{1}{2}$ whilst $|0011\rangle$ remains unaltered with amplitude $\frac{1}{2}$. We now have two eigenstates with $x=00$ and amplitude $\frac{1}{2}$: $|0000\rangle$ and $|0011\rangle$. So the probability of now observing one of these eigenstates is $\frac{1}{4} + \frac{1}{4} = \frac{1}{2}$. This is a better than classical algorithm. More generally, if $f(x)=11$ then we can consider the states $|x\ 11\rangle$ and $|x' f(x')\rangle$ (where x' is obtained from x by flipping the first bit) to obtain a similar result.

Furthermore, there would appear to be an obvious generalisation to n qubits: let the second negation on qubit 1 be controlled by all the qubits of $f(x)$. This is another example of an evolved circuit generalised by human analysis.¹⁰

6.3 Hitting the Physics: How many pulses does it take to make a CN ?

6.3.1 Scaling Down

All of the above work has used ‘basic’ gates to construct circuits and algorithms. However, what counts as ‘basic’ depends on your interests. In practice, even a simple two-qubit gate such as CN may require a multi-stage implementation. For example, Gershenfeld & Chuang [36] describe a Nuclear Magnetic Resonance (NMR) scheme for quantum computation based on ensembles of molecules. They show how radio frequency pulse sequences can be used to implement arbitrary single-qubit rotations and also the two-qubit CN gate. (This suffices for all computations.) In computer science terms, we would generally regard the usual basic gates as assembly language; the pulse sequence implementation is somewhat akin to a firmware instruction sequence. The series of rotations below is Gershenfeld & Chuang’s CN gate implementation (up to phase, which can be removed by further rotations):

$$CN_{12} = R_{y1}(-90)R_{z2}(-90)R_{z1}(-90)R_{z12}(180)R_{y1}(90)$$

$$CN_{12} = \frac{1}{2^{5/2}} \begin{pmatrix} 1 & -1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1-i & 0 & 0 & 0 \\ 0 & 1-i & 0 & 0 \\ 0 & 0 & 1+i & 0 \\ 0 & 0 & 0 & 1+i \end{pmatrix} \begin{pmatrix} 1-i & 0 & 0 & 0 \\ 0 & 1+i & 0 & 0 \\ 0 & 0 & 1-i & 0 \\ 0 & 0 & 0 & 1+i \end{pmatrix} \\ \times \begin{pmatrix} 1+i & 0 & 0 & 0 \\ 0 & 1-i & 0 & 0 \\ 0 & 0 & 1-i & 0 \\ 0 & 0 & 0 & 1+i \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & -1 & 1 \end{pmatrix} \\ CN_{12} = \sqrt{-i} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

There is a choice of pulse sequences (each implementing a rotation) to implement CN . Where there is choice, there is potential optimisation. Rethinam *et al* [76] use a basic genetic algorithm (bit string representation with single point crossover) to evolve pulse sequences to implement CN . The chromosome bit string is decoded as sequence of (rotational axis, angle) pairs. 9 bits are used for the angle

¹⁰ It must be acknowledged that the degree of improvement supplied by this generalisation decreases exponentially as n increases.

of rotation, allowing an accuracy one degree. They managed to evolve rotation sequences of length 3, more efficient than previously exhibited solutions. These are given below:

$$CN_{12} = R_{z_2}(270)R_{xz_{12}}(90)R_{x_1}(90)$$

$$CN_{12} = R_{x_1}(90)R_{z_2}(270)R_{xz_{12}}(90)$$

An important component of one solution to factor $N = 15$ comprises two successive CN gates (acting on qubits 1,2 and 2,3 respectively). The new CN implementation therefore improves the best achieved from 10 to 6 rotations. However, by composing the two operations and seeking a more direct implementation to this composed circuit a result was found using 5 rotations.

This field of work is significant. Efficiency effects integrity, since inefficient implementations will be more likely to suffer from environmental interference and faults due to the practicalities of carrying out operations. Thus, there is considerable merit in using evolutionary searches to derive excellent low-level implementations of gates.

Rotations have some angle θ as a parameter. Small changes in θ give rise to small changes overall: there is an element of natural continuity, which renders guided search particularly appropriate. CN is not a complex gate and the search space is very small compared with those of many problems attacked by evolutionary search. The search space may be too large for humans to derive optimal micro-circuits but it is clearly within the range of evolutionary search. The work of [76] is an important contribution.

6.3.2 Higher Level Basic Gates

What counts as a ‘basic gate’ is something of a moveable feast. Lukac *et al* [63] present a detailed investigation of how efficient low-level implementations can be evolved of some very well-known gates such as Toffoli, Fredkin, and Margolus gates. Their paper provides cost functions that are felt to be more realistic (in terms of physical realisation costs). Gate implementations by previous authors were successfully evolved together with several elegant new implementations. Local optimisation rules such as commutativity of certain operators (where the order in which gates are applied does not affect the result) are invoked to simplify and reduce costs. This work is a further (and clearly successful) demonstration that evolution, or heuristic search more generally, will find fruitful application across the spectrum of gate levels.

6.3.3 Location Matters

Most published circuits do not take into account where qubits physically reside during computation. Some current implementations may involve a line or small 2D lattice of qubits. In many implementations, two-qubit operations such as CN may only take place on neighbouring qubits, requiring qubit values to be progressively swapped until the required pair are neighbouring. These swaps are simply overheads to be optimised away [93]. There are also choices to be made as to the physical location where gates will be applied. Van Meter & Binkley [93] précis their current work on the allocation of qubits and gates to physical locations and its

solution by genetic algorithms and report that the approach produces “better layouts than hand-compiled programs for a 90-instruction program on 32 qubits”.

We believe that issues such as location and reducing the overheads arising due to features of specific hardware technologies will benefit further from applications of guided search.

6.3.4 Summary

We have seen how various evolutionary search techniques had been harnessed to explore algorithms and circuits expressed in terms of basic operations or gates. The search space was essentially ‘physics free’, or, more accurately, ‘implementation independent’. It is perfectly sensible for algorithm researchers to work in terms of *what* basic gates achieve rather than *how*. As noted earlier, progress in classical software development has been marked by a drive to ever-increasing levels of abstraction and there seems little reason to believe that quantum software should be different. However, the abstract concepts we manipulate must be implemented *in some manner* and these implementation issues must be addressed. The emerging work on implementation concerns highlights opportunities for evolutionary search. Efficient implementation at low levels has a major effect: everything is built on top of it. As Rethinam *et al* [76] point out, there is “enormous potential for simplifying the implementation of working quantum computers”.

6.4 Starting in the Right Place

As noted earlier, if you have a fully working circuit you may wish simply to optimise some non-functional property or properties. A common approach to such problems in computer science is simply to apply a succession of *functionality-preserving* transformations (also termed *semantics-preserving* transformations) to the artefact (e.g. program), each of which improves the property of interest. (To be precise, such transformations sometimes alter the functionality a little, but to an extent that does not matter for most purposes. For example, $(a*b)*c = a*(b*c)$ is a mathematical identity, but replacing an instance of the left hand phrase with the right hand one might give results of different precision if the variables are floating point numbers.)

In very recent work Maslov *et al* [64] derive efficient schemes for generating and storing identities for use in quantum sub-circuit substitution. They give examples of how repeated substitutions can provide significant optimisations. Circuit optimisation is a well-established concept in traditional hardware engineering and more recently in reversible circuit engineering. We believe that evolutionary search will find useful application to the (essentially) non-linear quantum circuits optimisation problem.

6.5 Communication, Teleportation and Entanglement

We can be fairly flexible as to what counts as an algorithm or program. As Clark & Jacob [17] point out, communication protocols can be thought of as programs implementing (sometimes unreliable) distributed computation. Quantum mechanics provides us with exciting opportunities to derive new protocols. One such protocol – *quantum teleportation* – has already captured the imagination of many. Several evolutionary search researchers have attempted to evolve circuits for it.

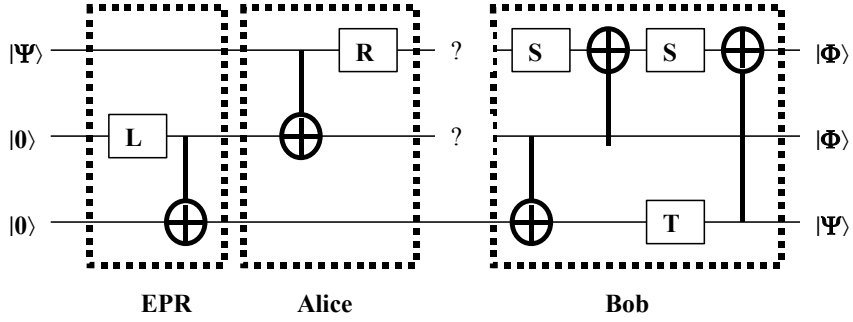


Figure 24. Teleportation circuit of Brassard [11].

6.5.1 Quantum Teleportation

Quantum teleportation is a means by which unknown quantum states can be transferred between locations using only classical channels and pre-existing entanglement. Brassard's original teleportation circuit [11] is shown in Figure 24.

The single qubit gates are defined by:

$$L = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \quad R = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \quad S = \begin{pmatrix} i & 0 \\ 0 & 1 \end{pmatrix} \quad T = \begin{pmatrix} -1 & 0 \\ 0 & -i \end{pmatrix}$$

As is usual in protocols, communication is between Alice and Bob.¹¹ The first two gates created a maximally entangled pair of qubits, the lower of which is sent to Bob. The other is sent to Alice. The next two gates on the send circuit serve to entangle all three qubits. The question marks denote measurements (giving rise to $|0\rangle$ or $|1\rangle$) by the sender Alice. The results are then communicated via classical channels to Bob who feeds them back in as the initial values of the top two qubits of his receive circuit.

Williams & Gray [97] use their list-based GP scheme to attack the design of the send and receive circuits. The work uses a rank based section scheme, to avoid premature domination of the population. There is a fair degree of optimisation sophistication in this work and it is unsurprising that this produces circuitry that improves on human design. The approach is able to produce a variety of send circuits of similar efficiency to the Brassard circuit and improved receive circuits. Furthermore the system used by Williams & Gray allows the user to restrict the choice of gates. A complete evolved circuit using only L , R and CN is shown in Figure 25.

The cost function used by Williams & Gray assumes that one knows the unitary transformation to be achieved by a sub-circuit. It measures deviation of the evolved unitary matrix S from the target matrix U :

$$f(S, U) = \sum_{i=1}^8 \sum_{j=1}^8 |U_{ij} - S_{ij}|$$

¹¹ Alice, Bob and Eve (who eavesdrops on communications between Alice and Bob) are the traditional actors in descriptions of classical secure communications protocols.

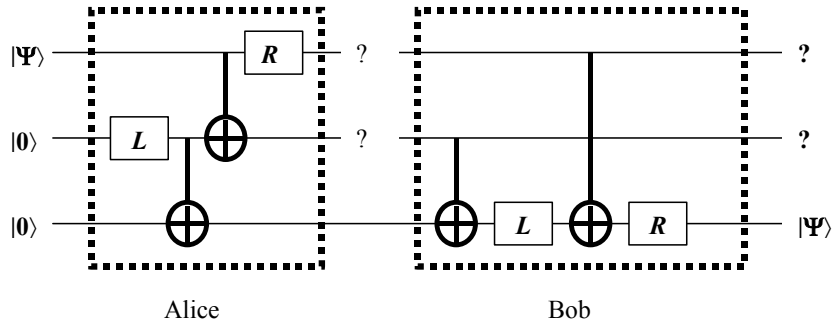


Figure 25. Evolved Teleportation circuit of Williams & Gray [97].

	0	1	2	3	0	1	2	3	0	1	2	3	
0	CN ₁₀	CN ₀₁				CN ₂₁	CN ₁₂		CN ₁₀	CN ₀₁	CN ₀₂		0
	CN ₁₀	CN ₀₁				CN ₂₁	CN ₁₂		CN ₁₀	CN ₂₁	CN ₁₂		1
	CN ₁₀	CN ₀₁				CN ₂₁	CN ₁₂		CN ₂₀				2
													3
1	L ₀	L ₁				L ₁	L ₂		L ₀	L ₁	L ₂		0
	L ₀	L ₁				L ₁	L ₂		L ₀	L ₁	L ₂		1
	L ₀	L ₁				L ₁	L ₂		L ₀	L ₁	L ₂		2
													3
2	R ₀	R ₁				R ₁	R ₂		R ₀	R ₁	R ₂		0
	R ₀	R ₁				R ₁	R ₂		R ₀	R ₁	R ₂		1
	R ₀	R ₁				R ₁	R ₂		R ₀	R ₁	R ₂		2
													3
3	separator				measurement								
	EPR Generation				Alice send				Bob receive				

Figure 26. Codon interpretation tables for the three stages.¹²

Yabuki & Iba [98] also address teleportation. They use a standard genetic algorithm approach, and they evolve a circuit in one go. They assume (and interpret everything in this context) that there are three stages (EPR pair preparation, send, and receive), that Alice can operate only on the first and second qubits, that measurement is allowed only once, and that the gates are restricted to $\{CN, L, R\}$. They use a fixed length chromosome comprising a sequence of three letter codons. The letters of the codon are chosen from $\{0,1,2,3\}$. In general, the first letter of a codon denotes the type of gate, the second identifies the qubits on which it operates. The third has a variable interpretation. The first codon starting with a 3 indicates the end of EPR generation and the start of Alice's send. The second such codon marks the partition between Alice and Bob's sections. The interpretation is based on tables for each section (Figure 26).

One chromosome for William & Gray's circuit (shown in Figure 25) is

112 | 231 | 001 | 331 | 132 | 012 | 221 | 302 | 001 | 100 | 002 | 201

EPR	Alice	Bob
-----	-------	-----

¹² For consistency within this review paper we have adopted the convention of C_{ij} denoting control qubit i and target qubit j . Yabuki & Iba [98] reverse this convention.

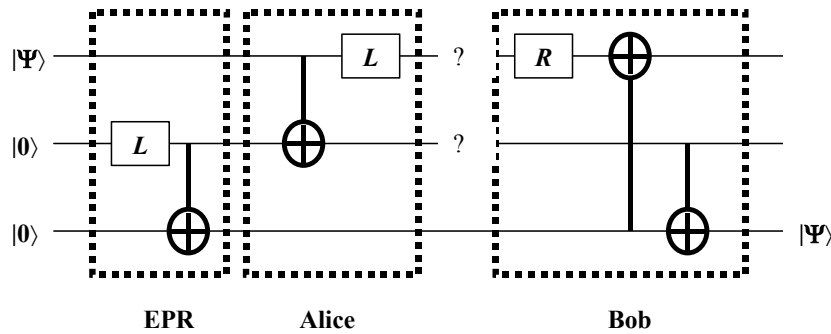


Figure 27. Yabuki & Iba's improved teleportation circuit.

The first codon 112 indicates the element in the major row 1, column indexed 1, minor row 2 of the EPR table, i.e. L_1 . 231 is ignored (empty element), 001 is CN_{10} and 331 marks the start of Alice's part. 132 is ignored, 012 as CN_{21} , 221 as L_2 . 302 is interpreted as Alice's measurement. Bob's four codons are interpreted as CN_{10} , L_0 , CN_{10} and R_0 . Yabuki and Iba evolved a simpler circuit, shown in Figure 27. The work also differs from that of Williams & Gray in that the evaluation function is based on three fitness cases based on how well the circuit actually teleports, i.e. how well it transfers the source qubit state exactly.

The reader may well be struck by just how *small* the original and the evolved quantum teleportation circuits really are. It suggests that truly novel quantum protocols could be well within reach of evolutionary search. The circuits (or sub-circuits) evolved were obtained in full knowledge of the structure of Brassard's original circuit. The concept of teleportation was known as was the structure of a solution. We believe that current successes suggest that the evolutionary search community should co-operate with the quantum information processing community to pose *new* and *unsolved* problems. Problem *solving* seems within our grasp; problem *finding* seems the immediate challenge.

6.5.2 Communication and Communication Resources

A variety of quantum protocols mix classical communication with the exploitation of entanglement. Teleportation is a high-profile example. Dense coding is another (where the communication of 1 classical bit of information coupled with a pre-existing entangled qubit pair allows 2 bits of classical information to be communicated between sender and receiver, see [70]). The tradeoffs between classical communication and quantum entanglement resources are improperly understood. Bennett has conjectured that a single use of any given two-particle transformation has a unique maximum power for entanglement or communication (for forward, backward, or two-way communication). Spector & Bernstein [87] report Smolin as suggesting one of the gates below (*SMOLIN*) as being capable of generating entanglement but not classical communication.

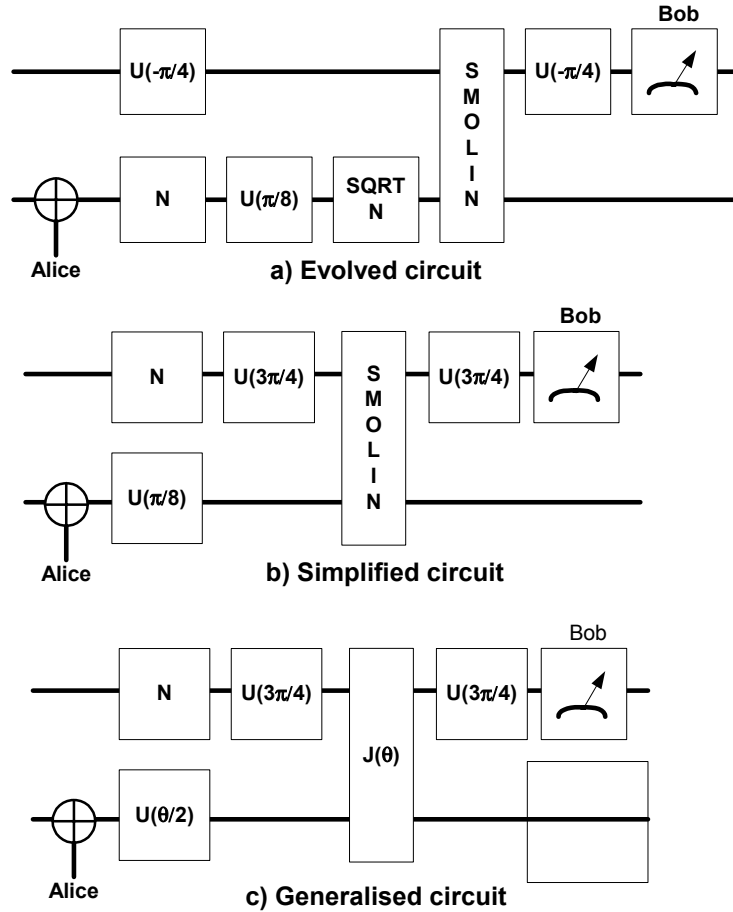


Figure 28. From evolved circuit to general idea.

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

$$J(\theta) = \begin{pmatrix} \cos \theta & 0 & 0 & \sin \theta \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \sin \theta & 0 & 0 & -\cos \theta \end{pmatrix} \quad BS(\theta) = \begin{pmatrix} \cos \theta & 0 & 0 & \sin \theta \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & 0 & -\cos \theta \end{pmatrix}$$

Spector & Bernstein [87] have evolved a circuit that allows one classical bit to be communicated per use of the *SMOLIN* gate. This was analysed and simplified, and subsequently generalised. The three stages of circuit derivation are shown in Figure 28. This is another excellent example of small evolved circuits acting as an intellectual spur to creativity in the field. We believe that such ‘concept seeding’ will be a major exploitation avenue for GP-based quantum work.

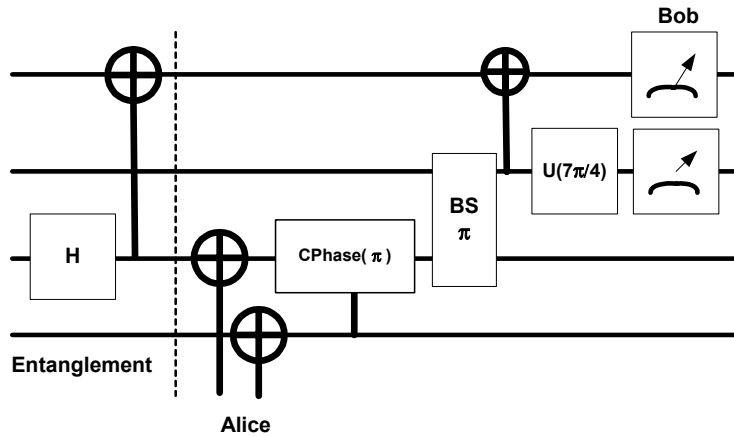


Figure 29. One-bit Prior Entanglement Allows Two Classical Bits Communication

Entanglement is often regarded as a ‘resource’. There are many measures of entanglement and it is not fully understood what entanglements can be achieved. Rubinstein [78] uses GP to evolve maximally entangled states for 3, 4 and 5 qubits. The system is using the same idea in each case; this is discernible on sight of the circuits produced. (As above, patterns can be recognised.)

Spector & Bernstein [87] demonstrate an evolved circuit that allows two bits of classical information to be communicated with one-bit of prior entanglement, as shown in Figure 29. We believe that the exploration of entanglement and communication along the lines of Spector & Bernstein’s work will prove a highly fruitful avenue for evolutionary search. Understanding the fundamental capabilities of quantum resources is a necessity.

6.6 Visualisation

Visualisation can often help to understand what is going on in complicated cases. Can we visualise the execution of a quantum algorithms as an aid to understanding?

A general single qubit state $|\Phi\rangle = a|0\rangle + b|1\rangle$ is characterised by the two complex amplitudes a and b . So at first sight this would appear to require a four-dimensional diagram. However, one of the dimensions reduces to an ignorable phase, leaving just three dimensions. The customary way to visualise this state is by using a *Bloch sphere*, taking the “north pole” to be $|0\rangle$ and the “south pole” to be $|1\rangle$. Superpositions lie elsewhere, but all on the surface of the sphere because of the normalisation condition $|a|^2 + |b|^2 = 1$. See, for example, [70, section 1.2] for more description. Note that, although the vectors $|0\rangle$ and $|1\rangle$ are *orthogonal*, they appear *anti-parallel* in the Bloch-sphere representation, which can sometimes cause confusion.

It is hard to visualise more than one qubit: an n -qubit state is characterised by 2^n complex numbers, or 2^{n+1} real numbers. Even losing one of these numbers as a phase factor is of little help.

The discussion above shows that small circuits of a few qubits are producing valuable results: sometimes new special purpose results, and sometimes small results that can then be generalised by a human. So it would seem worth considering the visualisation of just a small number of qubits, to further improve our intuition about quantum algorithms.

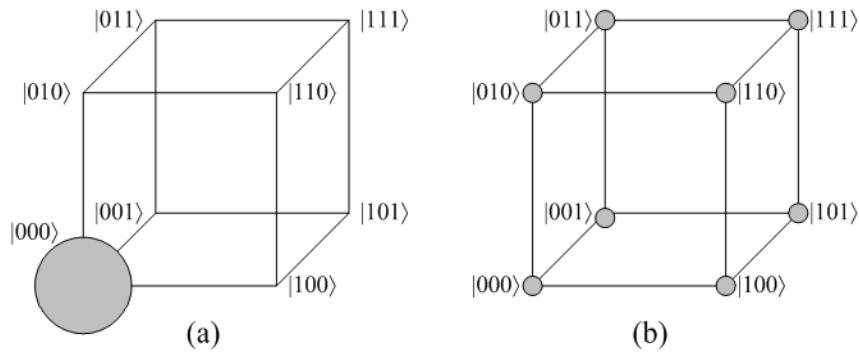


Figure 30. A 3-qubit Spector cube for (a) the state $|000\rangle$ (b) equal superposition of all states

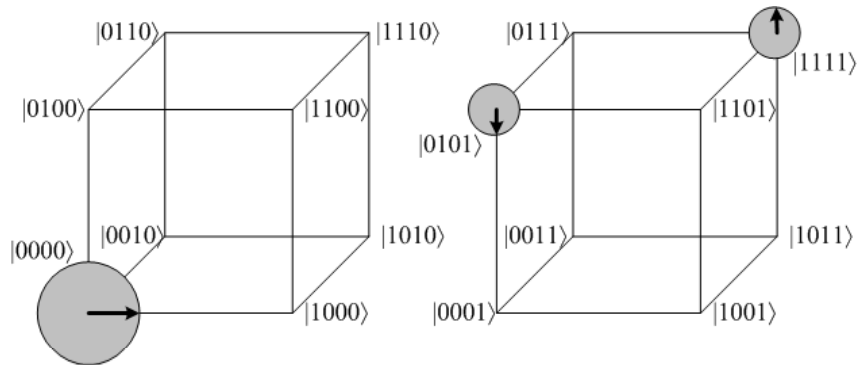


Figure 31. A 4-qubit complex Spector cube for the state $\sqrt{\frac{1}{6}}(2|0000\rangle - i|0101\rangle + i|1111\rangle)$

Spector [88, section 3.2] uses a *cube diagram* to represent the state of 3 qubits. Each corner of the cube represent one of the eight states, and a small disc drawn at each corner represents the amplitude of the respective state. In Spector’s diagrams, the size of the disc represents the absolute value of the amplitude, with a minus sign shown if the amplitude is negative: in Spector’s example quantum circuit, all amplitudes are real, which simplifies things. See Figure 30.

Spector shows the progress of Grover’s algorithm on 3 qubits as a sequence of cube diagrams [88, figs 3.4–3.13]. This sequence vividly shows the amplitudes initially being smeared out over all the states, and then coming together on the result states.

It seems worthwhile to explore this form of visualisation further, for more general cases. Complex amplitudes could be shown using a small vector in the complex plane at each cube corner (in Spector’s example, all such vectors are purely leftward or rightward pointing). Higher numbers of qubits could be shown. A 2D drawing of a 4D hypercube might still allow a sufficiently “natural” representation. This is topologically equivalent to a side-by-side pair of cubes. See Figure 31. This suggests that a pair of hypercubes (or a pair of pairs of cubes) could be used to represent a 5-qubit state.

6.7 Summary

The use of evolutionary computation to derive quantum artefacts has seen substantial progress in a short time. Since its origins in the late 1990s a considerable variety of problems have been attacked by evolutionary computation (and genetic algorithms and genetic programming in particular).

Even though the evolution of circuits and algorithms seems hard, we have seen several examples of novelty. Some work has pushed the frontiers of knowledge of quantum information processing, producing results of interest independent of the means of production.

Evolved artefacts have generally been ‘small’, which might raise worries about scalability. However, small artefacts are amenable to human analysis, and generalisations can be found.

Finally we note that Quantum Information Processing (QIP) is in its infancy. Understanding the possibilities and limits of what quantum systems and resources such as entanglement can offer will prove of major importance. It is encouraging to see that pieces of work are underway on fundamental problems of the topic.

7 Conclusions

The above discussion leads us to the following conclusions:

There is nothing much new! There is significant potential for discoveries. There are still very few fundamentally different quantum algorithms (however discovered).

The evolution of novel quantum artefacts is possible, but hard. The search landscape would appear to be extremely rugged and complicated, and our ability is currently limited to the evolution of small-scale artefacts.

Small may be beautiful. Heuristic searches for *implementations* of even ‘simple’ gates should prove beneficial. This is important in the same way that improved circuitry for adders and multipliers is important in classical computing.

From little acorns mighty oaks do grow. The human analysis of small artefacts can lead to general algorithms being discovered and we have seen several examples of this. We should aim to make best use of the abilities of highly gifted quantum researchers. (They have developed the subject this far.) The ability of heuristic searches to reach surprising results will most likely pique the interest in the quantum scientific community. Are we moving towards an era of GP-*assisted* discovery?

We should get back to basics. Work seems targeted at the evolution of specific circuits, algorithms and protocols. However, quantum mechanics itself is improperly understood. It is not known for example whether particular entanglements are achievable. There are various measures of entanglement and seeking to optimise these for particular circumstances has the potential to surprise and outperform the quantum mathematicians (for whom pen-and-paper analysis remains dominant.)

We need to use the power. All work in the area of evolving quantum artefacts seems to have been carried out using very modest hardware. But there is a significant trend to widen access to high-end computing power. (In the UK there is for example a substantial investment in ‘Grid computing’.) Furthermore, programmable hardware is now becoming very cheap, for example, racks of field programmable gate arrays (FPGAs) could be used to provide substantial simulation power. Koza has mapped the increasing success of GP to the rise in computing power since its emergence. We are now faced with a similar, if not greater, rise in sheer power. There would seem to be an opportunity to embrace the emerging availability of such resources.

The emergence of practical quantum computational facilities will enable even more interesting artefacts to be evolved.

8 The Future

This section is rather more speculative, and, based on the conclusions above, we discuss aspects of quantum computation that might show the greatest promise for meta-heuristic search techniques, and what work needs to be done to prepare the way.

8.1 Improving the simulation efficiency

The quantum search space is exponentially huge, which is why meta-heuristic search tools are being used. And the cost function evaluation is in its turn exponentially expensive to evaluate, because of the need to evaluate quantum algorithms on classical machines. This requires careful design if any but the most trivial quantum circuits are to be evolved. For example, selection strategies should be carefully chosen, and adding some noise may help with certain problems [58].

Attacking the expense of the cost function offers great potential improvement. There are certain techniques that improved the efficiency of classical simulation; see, for example, Viamontes *et al*'s QuIDD approach [94], and Massey *et al*'s 'row swapping' optimisation [65]. However, these provide significant speed-up only for circuits with a great deal of a certain kind of structure, unlike those that are generated by random evolutionary moves.

It is already common for cases of expensive cost functions (such as complicated finite element or fluid flow engineering applications) to use some kind of approximation in the early stages of the search, and to use the full cost function only towards the end, when the extra precision is necessary. See, for example [9]. The quantum circuit search space appears to be exceptionally rugged [57], which also makes search hard. A choice of approximate cost function that somehow 'smooths' the search space [18] may help to make search progress more effectively. The challenge with quantum circuits is to find suitable approximations and smoothings.

Eventually, when quantum computers become a reality, it will be possible to do a form of intrinsic evolution: evaluating the cost function directly on a quantum computer, thereby gaining exponential speedup over classical simulations. One should remember, however, that a classical simulation can calculate the *entire* probability distribution of the final state, not just provide the single observation that would be available intrinsically. This extra information available classically should be exploitable in current work.

8.2 New areas to explore

8.2.1 Global quantum properties

The most novel quantum algorithms (Grover's, Shor's) appear to be exploiting certain global properties of the quantum state space, rather than acting on localised qubits. What other applications would benefit from this kind of solution? Should we modify search cost functions to reward this kind of solution?

8.2.2 Probabilistic results

In addition to the global properties, quantum algorithms are intrinsically probabilistic, and this feature should be exploited more. There is a rich area of classical probabilistic algorithms (see for example [69]) that should be carefully examined for quantum possibilities and inspiration.

There are two different interpretations of what “correct with a probability of $p\%$ ” can mean: (1) for a certain $p\%$ of its inputs, the circuit gives the right answer every time (2) for all of its inputs, the circuit that gives the right answer with probability $p\%$ each time. Each interpretation leads to different kinds of cost functions, and different kinds of quantum algorithms.

8.2.3 Quantum protocols

The evolutionary search work has tended to concentrate on quantum algorithms. Quantum protocols also offer a rich area for search, and evolutionary search has been used to some degree to explore quantum teleportation and dense coding, as noted above.

Evolutionary techniques have been used with success to discover efficient classical communication protocols, for example [17], using the classical protocol reasoning BAN logic [15]. Can this approach be extended to quantum protocols? How do we need to extend or change the logics to handle quantum protocols?

8.2.4 Other computational models

Most of the work currently evolves quantum circuits of qubits. There are other models of quantum computation that could be explored.

It is not necessary to restrict the quantum values to be binary qubits: qudits, d -dimensional quantum values, are worth investigating. It is not even necessary to restrict the quantum values to be discrete: continuous quantum algorithms exist [13], and might form a ‘smoother’ search space.

Non-circuit based models of quantum computation, such as quantum cellular automata [14] [81], and measurement-based quantum machines [73], could prove to have more tractable search spaces.

8.3 Other applications

In addition to searching for particular quantum circuits, meta-heuristic search can be used for other applications in the quantum domain.

8.3.1 Searching for quantum solutions

Meta-heuristic search has been successfully used to find counter-examples to conjectures in classical domains, such as classical cryptography [18]. There are many conjectures in quantum computation and quantum information theory (such as bounds on entanglement, and channel capacities) that offer suitable targets to meta-heuristic searches for counterexamples. For example, Spector & Bernstein [87] use genetic programming to discover new bounds on quantum communication.

Certain quantum states are particularly ‘interesting’, for example, they are highly entangled. These states are known for small quantum systems (2 or 3 particles), but not for larger ones. Search against a suitable cost function can be used to find these larger interesting states [12].

8.3.2 Quantum random walks

Quantum random walks [49] are quantum analogues of classical random walks, with very different properties. For example, the probability distribution for a classical random walk is binomial, with a peak at the origin, and a width $O(\sqrt{n})$ after n steps, whilst the probability distribution for a quantum random walk is strongly peaked at

$O(\pm n/\sqrt{2})$. Childs *et al* [16] use a quantum random walk algorithm to find a path through a graph exponentially faster than classically. Quantum random walks can be used as the basis of efficient search algorithms [82].

8.3.3 Quantum genetic algorithms

Grover's algorithm uses quantum effects to perform searches more efficiently than can classical algorithms. Quantum random walks offer potential exponential speedup of certain searches. Quantum effects exploit global properties of the state space; genetic algorithms, as explained by the schema theorem, perform an implicitly parallel search over global hyperplanes in the search space [46].

Can these two global, parallel processes be combined to produce a quantum genetic algorithm? The quantum "inspiration" may simply be the idea of superposition [41] [42]. More detailed work by Rylander *et al* [80] suggests that getting a fruitful combination is non-trivial. It seems we will need yet further intuition priming to understand how to combine quantum inspiration with classical results.

9 Bibliography

1. Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, Shlomo Angel. *A Pattern Language: towns, buildings, construction*. OUP, 1977
2. Thomas Bäck. *Evolutionary Algorithms in Theory and Practice*. OUP, 1996
3. Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, Frank D. Francone. *Genetic Programming, An Introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann, 1998
4. Howard Barnum, Herbert J. Bernstein, Lee Spector. A quantum circuit for OR. quant-ph/990756
5. H. Barnum, H. J. Bernstein, L. Spector. Quantum circuits for OR and AND of ORs. *J. Physics A*, **33**(45):8047–8057, Nov 2000
6. Kent Beck. *Smalltalk Best Practice Patterns*. Prentice Hall, 1997
7. Paul Benioff. The computer as a physical system: a microscopic quantum mechanical Hamiltonian model of computers as represented by Turing Machines. *J. Stat. Phys.* **22**, 563–591, 1980
8. Paul Benioff. Quantum mechanical Hamiltonian models of Turing machines that dissipate no energy. *Phys. Rev. Lett.* **48**, 1581–1585, 1982
9. Andrew J. Booker, J. E. Dennis Jr., Paul D. Frank, David B. Serafini, Virginia Torczon, Michael W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural Optimization* **17**(1):1–13, 1999
10. Dirk Boumeister, Artur Ekert, Anton Zeilinger, eds. *The Physics of Quantum Information: Quantum Cryptography, Quantum Teleportation, Quantum Computation*. Springer, 2000
11. G. Brassard. Teleportation as Quantum Computation. In *Proc. 4th Workshop on Physics and Computation, New England Complex Systems Institute 1996*. Also as quant-ph/9605035, 1996.
12. Iain D. K. Brown, Susan Stepney, Anthony Sudbery, Samuel L. Braunstein. Searching for entanglement. (submitted)
13. S. L. Braunstein, A. K. Pati, eds. *Quantum Information with Continuous Variables*. Springer, 2003
14. Gavin K. Brennan, Jamie E. Williams. Entanglement dynamics in one-dimensional quantum cellular automata. *Phys. Rev. A* **68**, 042311, 2003

15. Michael Burrows, Martín Abadi, Roger Needham. A logic of authentication. *SRC Research report 39*. Feb 1989. <http://gatekeeper.research.compaq.com/pub/DEC/SRC/research-reports/abstracts/src-rr-039.html>
16. Andrew M. Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, Daniel A. Spielman. Exponential algorithmic speedup by quantum walk. In *Proc. 35th ACM Symp. Theory of Computing (STOC 2003)*, pp. 59–68. ACM 2003
17. John A. Clark, Jeremy L Jacob. Protocols are Programs Too: the Meta-heuristic Search for Security Protocols. *Information & Software Technology* **43**(14): 891–904, Dec 2001
18. John A. Clark, Jeremy L. Jacob, Susan Stepney. Searching for cost functions. In *CEC 2004*, pp 1517–1524. IEEE 2004.
19. J. M. Daida, Adam M. Hilss, David J. Ward, Stephen L. Long. Visualizing tree structures in Genetic Programming. *GECCO 2003*, pp 1652–1664. LNCS 2724, Springer, 2003
20. J. M. Daida, Adam M. Hilss. Identifying structural mechanisms in standard Genetic Programming. *GECCO 2003*, pp 1639–1651. LNCS 2724, Springer, 2003
21. J. M. Daida, Hsiao-wei Li, Ricky Tang, Adam M. Hilss. What makes a problem GP-hard? Validating a hypothesis of structural causes. *GECCO 2003*, pp 1665–1677. LNCS 2724, Springer, 2003
22. Charles Darwin. *On the Origin of Species by means of natural selection, or the preservation of favoured races in the struggle for life*. Murray, 1859
23. Erasmus Darwin. *Zoonomia*, book II, 1795
24. Richard Dawkins. *The Blind Watchmaker*. Longman, 1986
25. Leandro de Castro, Jonathan Timmis. *Artificial Immune Systems: a new computational intelligence approach*. Springer, 2002
26. David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. R. Soc. Lond. A* **400**, 97–117, 1985
27. David Deutsch, Richard Josza. Rapid solution of problems by quantum computation. *Proc. R. Soc. Lond. A* **439**, 553–558, 1992
28. David Deutsch. *The Fabric of Reality*. Penguin, 1997
29. Bryce S. DeWitt, R. Neill Graham, eds. *The Many-Worlds Interpretation of Quantum Mechanics*. Princeton University Press, 1973
30. D. Divincenzo, J. Smolin. Results on two-bit gate design on quantum computers. In W. Porod, G. Frazier, eds, *Proc. 2nd Workshop on Physics and Computation (PhysComp '94)*, pp 14–23. IEEE, 1994
31. Hugh Everett III. “Relative State” Formulation of Quantum Mechanics. *Rev. Mod. Phys.* **29**(3): 454–462, July 1957
32. Richard P. Feynman. Simulating physics with computers. *Int. J. Theor. Phys.* **21**(6/7):467–488, 1982
33. Lawrence J. Fogel. *Biotechnology: Concepts and Applications*. Prentice Hall, 1963
34. Martin Fowler. *Analysis Patterns: reusable object models*. Addison Wesley, 1997
35. Erich Gamma, Richard Halm, Ralph E. Johnson, John Vlissides. *Design Patterns: elements of reusable object-oriented software*. Addison Wesley, 1995
36. Neil A. Gershenfeld, Isaac L. Chuang. Bulk Spin-Resonance Quantum Computing. *Science* **275**, 350–356, January 1997
37. David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
38. David E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer, 2002
39. Phil Gossett. Quantum Carry-Save Arithmetic. quant-ph/9808061, 1998
40. Lov K. Grover. A fast quantum mechanical algorithm for database search. *Proc. 28th Ann. ACM Symp. Theory of Computing (STOC)*, pp 212–219, 1996
41. Kuk-Hyun Han, Jong-Hwan Kim. Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *IEEE Trans. Evoln. Comp.* **6**:580–593, Dec 2002

42. Kuk-Hyun Han, Jong-Hwan Kim. Quantum-Inspired Evolutionary Algorithms with a new termination criterion, H_ϵ Gate, and Two-Phase Scheme. *IEEE Trans. Evoln. Comp.* **8**(2):156–169, April 2004
43. G. R. Harik, F. G. Lobo, David E. Goldberg. The compact genetic algorithm. *IEEE Trans. Evoln. Comp.* **3**(4) 287–297 1999.
44. W. Daniel Hillis. Co-Evolving parasites improve simulated evolution as an optimization procedure. *Physica D* **42** 228–234, 1990.
45. Tad Hogg. Solving Highly Constrained Search Problems with Quantum Computers. *J. Artificial Intelligence Research* **10**:39–66, 1999
46. John H. Holland. Genetic Algorithms and the Optimum Allocation of Trials. *SIAM J. Comp.* **2**(2):88–105 1973
47. John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
48. Wolfgang Kantschik, Wolfgang Banzhaf. Linear Tree GP and its comparison with other GP structures. In *EuroGP 2001*, pp 302–312. LNCS 2038, Springer, 2001
49. Julia Kempe. Quantum random walks – an introductory overview. *Contemporary Physics* **44**(4):307–327, 2003
50. Kenneth E. Kinneer Jr, ed. *Advances in Genetic Programming*. MIT Press, 1994
51. John R. Koza. *Genetic Programming: on the programming of computers by means of natural selection*. MIT Press, 1992
52. John R. Koza. *Genetic Programming II: automatic discovery of reusable programs*. MIT Press, 1994
53. John R. Koza, Forrest H. Bennett III, David Andre, Martin A. Keane. *Genetic Programming III: Darwinian invention and problem solving*. Morgan Kaufmann, 1999
54. John R. Koza, Martin A. Keane, Matthew J. Streeter, William Myrdlowec, Jessen Yu, Guido Lanza. *Genetic Programming IV: routine human-competitive machine intelligence*. Kluwer, 2003
55. Sanjeev Kumar, Peter J. Bentley, eds. *On Growth, Form and Computers*. Elsevier, 2003
56. André Leier, Wolfgang Banzhaf. Evolving Hogg’s Quantum Algorithm Using Linear-Tree GP. In *GECCO 2003*, pp 390–400. LNCS 2723, Springer, 2003
57. André Leier, Wolfgang Banzhaf. Exploring the search space of quantum programs. In *CEC 2003*, pp 170–177. IEEE Press, 2003
58. André Leier, Wolfgang Banzhaf. Comparison of Selection Strategies for Evolutionary Quantum Circuit Design. In *GECCO 2004*, pp 557–568. LNCS 3103, Springer, 2004
59. Hoi-Kwok Lo, Sandu Popescu, Tim Spiller, eds. *Introduction to Quantum Computation and Information*. World Scientific Publishing, 1998
60. Chris Lomont. Quantum circuit identities. quant-ph/0307111, 2003
61. Michael A. Lones. *Enzyme Genetic Programming: Modelling Biological Evolvability in Genetic Programming*. PhD thesis, Department of Electronics, University of York, 2003
62. Michael A. Lones, Andy M. Tyrrell. Enzyme Genetic Programming. In *CEC 2001*, pp 1183–1190. IEEE Press, 2001
63. Martin Lukac, Marek Perkowski, Hilton Goi, Mkhail Pivtoraiko, Chung Hyo Yu, Kyusik Chung, Hyunkoo Jee, Byung-guk Kim, Yong-Duk Kim. Evolutionary Approach to Quantum Reversible Circuits Synthesis. *Artificial Intelligence Review* **20**:361–417, Kluwer, 2003.
64. D. Maslov, C. Young, D. M. Miller, G. W. Dueck. Quantum Circuit Simplification using Templates. *2005 Conference on Design and Test Europe (DATE)*, submitted Sept. 12, 2004. From the web site of D. M. Miller, <http://www.cs.uvic.ca/~mmiller/publications/DATE0409.pdf>
65. Paul Massey, John A. Clark, Susan Stepney. Evolving quantum circuits and programs through genetic programming. In *GECCO 2004*. LNCS 3103, 569–580. Springer, 2004
66. N. David Mermin. From classical state-swapping to quantum teleportation. quant-ph/ 0105177 v4, 2002
67. Zbigniew Michalewicz, David B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2000
68. Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996

69. Rajeev Motwani, Prabhakar Raghavan. *Randomized Algorithms*. CUP, 1995
70. Michael A. Nielsen, Isaac L. Chuang. *Quantum Computation and Quantum Information*. CUP, 2000
71. Allen E. Nix, Michael D. Vose. Modeling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence* **5**(1): 79–88, 1992
72. Michael O’Neill, Conor Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer, 2003
73. Simon Perdrix, Philippe Jorrand. Measurement-Based Quantum Turing Machines and their Universality. quant-ph/0404146, 2004
74. Joseph Polchinski. Weinberg’s nonlinear quantum mechanics and the Einstein-Podolsky-Rosen paradox. *Phys. Rev. Lett.* **66** 397–400, 1991
75. Ingo Rechenberg. *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Frommann-Holzboog, 1963
76. Manoj Jesu Rethinam, Amil Kumar Javali, E.C. Behrman, J.E. Steck, S. R. Skinner. A genetic algorithm for finding pulse sequences for NMR quantum computing. quant-ph/0404170 v1, April 2004
77. Eleanor G. Rieffel, Wolfgang Polak. An Introduction to Quantum Computing for Non-Physicists. *ACM Computing Surveys* **32**(3):300–335, September 2000
78. Ben I. P. Rubinstein. Evolving Quantum Circuits Using Genetic Programming. In *CEC 2001*, pp 144–151, IEEE Press, 2001
79. Conor Ryan, J. J. Collins, Michael O’Neill. Grammatical Evolution: Evolving Programs for an Arbitrary Language. In *EuroGP’98*, pp 83–96. LNCS 1391, Springer, 1998
80. Bart Rylander, Terry Soule, James Foster, Jim Alves-Foss. Quantum Evolutionary Programming. *GECCO 2001*, pp 1005–1011. Morgan Kaufman, 2001
81. B. Schumacher, R.F. Werner. Reversible quantum cellular automata. quant-ph/0405174, 2004
82. Neil Shenvi, Julia Kempe, K. Birgitta Whaley. A Quantum Random Walk Search Algorithm. *Phys. Rev. A* **67**(5) 052307, 2003
83. Peter Shor. Algorithms for quantum computation: discrete log and factoring. *Proc. 35th Ann. Symp. Foundations of Computer Science*, pp 124–134. IEEE, 1994
84. L. Spector, H. Barnum, H. J. Bernstein, N. Swamy. *Genetic Programming for Quantum Computers*. In *Genetic Programming 1998*, pp. 365–374. Morgan Kaufman, 1998
85. L. Spector, H. Barnum, H. J. Bernstein, N. Swamy. *Finding a Better-than-Classical Quantum AND/OR Algorithm using Genetic Programming*. In *CEC 1999*, pp. 2239–2246. IEEE, 1999
86. Lee Spector, Howard Barnum, Herbert J Bernstein, Nikhil Swamy. *Quantum Computing Applications of Genetic Programming*. In L. Spector, W. B. Langdon, U.-M. O’Reilly, P. J. Angeleine, eds, *Advances in Genetic Programming 3*, chapter 7, pp 135–160. MIT Press, 1999
87. L. Spector, H. J. Bernstein. Communication Capacities of some Quantum Gates, discovered in part through Genetic Programming. In *Proc. 6th Int. Conf. Quantum Communication, Measurement, and Computing (QCMC)*, pp. 500–503. Rinton Press, 2003
88. Lee Spector. *Automatic Quantum Computer Programming: a genetic programming approach*. Kluwer, 2004.
89. Guy L. Steele. *Common Lisp the Language, 2nd edition*. Digital Press, 1990
90. Susan Stepney, Fiona Polack, Ian Toyn. Patterns to Guide Practical Refactoring: examples targetting promotion in Z. In *ZB2003*, pp 20–39. LNCS 2651, Springer, 2003
91. Susan Stepney, Samuel L. Braunstein, John A. Clark, Andy Tyrrell, Andrew Adamatzky, Robert E. Smith, Tom Addis, Colin Johnson, Jonathan Timmis, Peter Welch, Robin Milner, Derek Partridge. Journeys in Non-Classical Computation II: Initial Journeys and Waypoints. *Int. J. Parallel, Emergent and Distributed Systems* 2005 (to appear)
92. Adrian Thompson, Paul Layzell. Analysis of unconventional evolved electronics. *Comm. ACM* **42**(4) 71-79, 1999.
93. Rodney Van Meter, Kevin Binkley. Compiling Quantum programs Using Genetic Algorithms. In *The Wild and Crazy Idea Session IV*, abstracts, part of *11th Intl. Conf. Architectural Support for Programming Languages and Operating Systems*, October 2004

94. George F. Viamontes, Manoj Rajagopalan, Igor L. Markov, John P. Hayes. Gate-Level Simulation of Quantum Circuits. quant-ph/0208003, 2002
95. M. Weiser. The Computer for the 21st Century. *Scientific American* **265**(3): 66–75, Sept 1991
96. Colin P. Williams, Scott H. Clearwater. *Explorations in Quantum Computing*. Springer, 1997
97. Colin P. Williams, A. G. Gray. Automated Design of Quantum Circuits. In *Quantum Computing and Communications: First NASA Conference, QCQC'98*, pp 113–125. LNCS 1509, Springer, 1999.
98. Taro Yabuki, Hotoshi Iba. *Genetic algorithms for quantum circuit design – evolving a simpler teleportation circuit*. In *Late Breaking Papers at GECCO 2000*, pp. 425–430, August 2000.
99. Ricardo Salem Zebulum, Marco Aurelio C. Pacheco, Marley Maria B. R. Vellasco. *Evolutionary Electronics: automatic design of electronic circuits and systems by genetic algorithms*. CRC Press, 2002