# Searching for Quantum Programs and Quantum Protocols: a review*

Susan Stepney, John A. Clark
Department of Computer Science,
University of York, UK, YO10 5DD

October 9, 2007

**Abstract**

We review how computational search techniques inspired by biological evolution have been used to discover quantum circuits and quantum algorithms. We discuss issues in representing quantum artefacts in a form suitable for evolutionary search, various quantum artefacts that have been discovered through evolutionary search, and future prospects for this approach.

**keywords:** quantum algorithms, heuristic search, evolutionary algorithms

# Contents



*This review is an updated and revised version of [1]

# 1 Introduction

The modern computer pervades most aspects of our lives. In a short space of time we have moved from computers that filled many rooms to Weiser's ubiquitous computing vision [2], where computers disappear into the fabric of everyday objects 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 [3]. Subsequently, a slower but still impressive rate of doubling approximately every 18 months

has been observed. (This is now generally referred to as Moore's Law.) Early home computers had only a few kilobytes of memory; today, commercially available 'memory sticks' 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.

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 quantum mechanics' development, as model of behaviour of small-scale phenomena, it 'works'. Although the 'meaning' of quantum mechanics is still hotly debated, 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, Richard Feynman noted that harnessing quantum phenomena of matter could allow complicated systems to be simulated effectively [4]. 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 [5, 6].

In 1985 David Deutsch showed how the classical computation model (the Turing machine) could be simulated using quantum mechanical properties of matter [7]. Since the Turing Machine is felt to capture what is meant by 'computation' (see the discussion in [8, 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 [7] (concerning the single bit XOR or parity problem).

The biggest practical impetus came in 1994 when Peter Shor demonstrated a quantum analogue of the Discrete Fourier Transform [9]. 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'.

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 §B) but there are fundamentally few distinct quantum algorithms. In some ways novel application development seems to have stalled, as Williams & Clearwater [11] note:

> 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.

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. Here, we review how automated search techniques inspired by biological systems can be used to uncover new quantum circuits and algorithms.

We review the use of *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.

We provide background material in the appendices. §A has background on quantum computation and quantum circuits, and §B overviews some known quantum algorithms. §C overviews metaheuristic search terminology, and various kinds of evolutionary algorithms.

The body of the review comprises two main sections. In §2 we consider issues to do with implementing evolutionary search for quantum circuits and

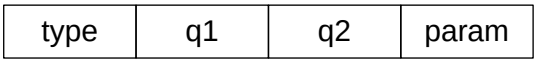



Figure 1: gate template

protocols. Then in §3 we review the results to date of such searches. In the final section we discuss future possibilities.

# 2 Implementation issues

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

## 2.1 Representation of candidate solutions

### 2.1.1 Direct encodings

A quantum circuit can be represented as list of gates. The order in which gates appear in the list is the order in which the corresponding unitary transformations are applied (see §A).

A *gate template* is a list of slots, where each slot can be instantiated to attribute values to give a gate. Figure 1 shows a template with a slot for the type of gate ($H$, $N$, $CN$, 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 appropriately for each gate type, or ignored if not needed. For example: a NOT gate acting on qubit 3 is represented as $(N, 3, *, *)$, where '$*$' means the value in the slot is ignored; a controlled-NOT gate with control qubit 3 and target qubit 1 is represented as $(CN, 3, 1, *)$; a single qubit rotation $U(\pi)$ on qubit 5 is represented by $(U, 5, *, 3.14159)$.

The most basic low level representation of a list of gates is as a bit string. (Bit string representations are common in genetic algorithm applications.) Figure 2 shows how sub-sequences of bits might map onto gate slots. So a string of bits can be decoded as a list of gates, and hence as a quantum circuit.

Such encodings are not without their problems. Consider a search over 5 gate types, thus requiring at least three bits to represent. But three bits can represent 8 gate types. We may ensure that an initial population has type fields with bit values of 000, 001, 010, 011, 100 (that is, only between 0 and 4), but simple crossover and mutation operations can produce 101, 110, and 111. These values need to be interpreted as valid gates in some way. Interpreting

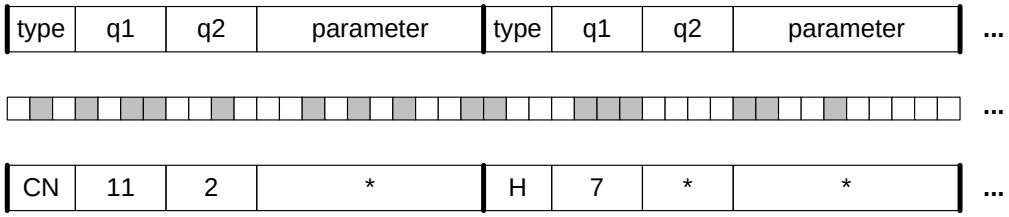



Figure 2: The gate template (top line) is used to interpret a bitstring (middle line) as a list of tuples representing specific gates (bottom line)

the value modulo 5 produces an acceptable type index, e.g. 110 denotes 6, and 6 mod 5 is equal to 1, and so 110 would represent gate type 1. With this scheme, 0 is represented by $(000, 101)$, 1 by $(001, 110)$, 2 by $(010, 111)$, but 3 and 4 have a single representation of 011 and 100 respectively. Thus, some elements of the space are over-represented, possibly biasing certain types of search. Other interpretion schemes have their own biases.

Some researchers have used similar simple bit string representations with genetic algorithms. Although such representations are considered unsophisticated by the wider evolutionary computation community, their application is not without some success (for example, see the gate implementation work described in §3.5, and the teleportation circuits described in §3.6), although this success may be *in spite of* the representation used.

It is, of course, possible to work directly with character, integer and real values.

### 2.1.2 Linear list encodings

*Linear genetic programming* (see §C.5) can be used to encode variable length lists of quantum gates. This provides a natural representation and powerful approach for the evolution of quantum circuitry.

Williams & Gray's GP approach [12] exemplifies the flexibility afforded by such schemes, with a variety of evolution operators provided: mutation, substitution, crossover, transposition, insertion and deletion. (They do not incorporate features that allow instructions to be skipped.)

Spector *et al* [13] 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 im-

plemented 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 and Gray [12] . 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.

### 2.1.3 Second order encodings

With some linear list variants, the representations code for specific solutions. 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 *scaleable* artefacts. A feature of scaleable human-developed artefacts is the use of *structure*. For example, a classical adder circuit comprises a connected series of single bit adders. A 12-bit adder looks a lot like a 10-bit adder; both are 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 also have significant structuring mechanisms: if-then-else; for-loops, while-loops; functions; procedures etc. Functions and procedure provide high-level reusable building blocks, and are usually parameterisable. We would like similar facilities to be provided for the evolution of quantum algorithms.

Structure additionally aids human understanding, because it captures an intellectual and communicable idea. Human understanding of evolved artefacts may be an important goal for evolutionary search in the quantum domain: it will enable further generalisation of found solutions (§3.6.2), and deeper understanding of the uncovered mechanisms.

Structure 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 (§3.4).

### 2.1.4 Spector et al's traditional GP tree encoding

The early GP work by Spector *et al* (§3.1) uses a second order 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 standard gates to the circuit. Examples are:

- H-GATE: appends a Hadamard gate to the the circuit. It has one parameter, coerced to a valid qubit index.
- U-THETA-GATE: appends a rotation gate. It has two parameters: the first is coerced to a valid qubit index, the second is an angle in radians.
- CNOT-GATE: appends a controlled-NOT gate. It has two parameters, coerced to form valid source and target qubit indexes.

These functions return the value of their first argument as their result.

A variety of helpful support functions are provided (e.g. mathematical operations such as $+$, $-$, $*$ etc.). Iteration constructs are provided, such as:

- ITERATE. This takes two parameters: the second is a subprogram body; the first, coerced to a non-negative integer, is the number of 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.

These iteration constructs allow the system to evolve *scalable* algorithms, parameterised to be used on systems of different sizes.

The language has a general LISP flavour and representation. Consider the program in figure 3a; when executed this will produce the result in figure 3b, which describes the circuit shown in figure 3c.

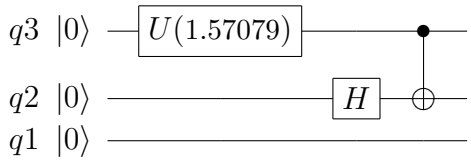
A typical problem with basic GP approaches is that they are weakly typed. The coercion of returned values to parameters is somewhat unconvincing and represents a significant potential restriction on the programs that can practically be evolved. Spector *et al* [13] also note that tree representation 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.

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

(a) example program

```
U-THETA   3   1.57079   // PI = 3.14150 is coerced to 3
H         2             // 1 + PI/2 = 2.57079 is coerced to 2
C-NOT     3   2         // PI = 3.14150 is coerced to 3
                        // 2.57079 is coerced to 2
```

(b) result of execution




(c) described circuit

Figure 3: Second order encoding of a quantum circuit

### 2.1.5 Leier and Banzhaf's Linear Tree GP representations

Kantschik & Banzhaf [14] 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 4). 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 [15] 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.

### 2.1.6 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 book [16] is given over to explaining the basics of the underlying technological tools. PUSH is a Lisp-like programming language with very simple syntax:

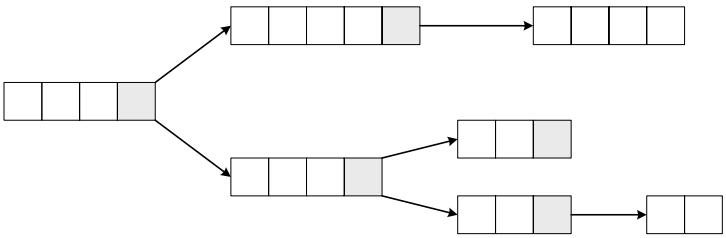


Figure 4: Linear tree GP representation. White boxes represent program nodes; grey boxes represent branching nodes.

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

There are stacks for different data type operations (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 **then** push it onto the appropriate stack

**else** $\{P$ is a list$\}$ sequentially execute each program in the list

For example, execution of the program

```
( ( 5 4 INTEGER.+ ) ( 2.0 2.0 FLOAT.* ) )
```

causes 5 and then 4 to be pushed on the integer stack; the 4 and 5 to be popped from the integer stack and added; the result (9) placed back on the integer stack; then 2.0 and 2.0 are pushed on the float stack; they are popped and multiplied; the result (4.0) is pushed on the float stack. See [16] 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 [16] (including visualisation of simulations) collectively form a cohesive quantum genetic programming research suite. The underlying simplicity of the supporting technology is striking.

## 2.2 Evaluating candidate solutions

Evaluation functions (cost or fitness functions) define what it means to be a desirable solution to a problem, and provide *guidance* to the search process to find more desirable solutions. Williams & Gray [12] note: "We regard the most sensible evaluation measure as an open question". This remains the case at the time of writing this review (2007).

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

### 2.2.1 Deviation from target matrix

In their approach Williams & Gray [12] 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

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

This is an intuitively appealing function and it is applied with some success (see later). The choice of magnitude of differences is not definitive. Williams & Gray [12] use $R = 1$; DiVincenzo & Smolin [17] use $R = 2$. Non-integral values or $R$ might prove useful: Clark *et al* [18] demonstrate the sensitivity of some (cryptographic) problems to exponent choice.

Lukac *et al* [19] 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 here) and circuit cost (see §2.2.3).

### 2.2.2 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 [20] 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* [21] use a cost function given by

$$f = \sum_i \left|\left| V_{T_i} - V_{R_i} \right|\right| \tag{2}$$

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 magnify 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*. (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.)

Spector *et al*'s work [22, 23, 24, 13] has a probabilistic notion of success and uses a fitness function that captures functional correctness but also aspects of efficiency. It has the form

$$f = hits + correctness + efficiency \tag{3}$$

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

$$correctness = \frac{\sum_{i=1}^{n} \max(0, error_i - 0.48)}{\max(hits, 1)} \tag{4}$$

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. simply improving from a 55% success rate to a 60% success rate), errors smaller than 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; so the 'pure' *correctness* term is divided by *hits* (unless $hits < 1$).

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* [21] for probabilistic circuits. More general fitness functions (but using many of the same concepts) can be found in Spector's book [16]. Spector *et al* [13] note that the fitness function evolved as their work reported progressed. Further fitness function details can be found in [23].

### 2.2.3 Resource usage

Not all search problems start from no idea of the solution. Given a working circuit, you may wish to improve it in some way. Compilers for traditional programming languages generally have an optimiser, which applies a series of *semantics-preserving transformations* to obtain a program with some improved 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, [25]).

Maslov *et al* [26] discuss *linear cost circuit metrics* (a simple weighted gate count) and *non-linear circuit cost metrics* (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. (See also §2.3.2 and §3.5.6).

## 2.3 Other issues

### 2.3.1 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 [27] 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 [27] 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.

### 2.3.2 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 optimisation by semantics-preserving transformations. This leads to a direct way for highly efficient circuits to be created from inefficient, but functionally correct, ones.

### 2.3.3 Simulation issues

Cost function evaluation currently requires simulation of a quantum computation on a classical hardware platform. Simulation efficiency is of major practical importance.

Spector [16] acknowledges such issues. Consider how a unitary transform should be stored. The most straighforward approach is to store its $2^{2n}$ element matrix, but this becomes unmanageable as the number of qubits in a system grows. A 15-qubit system would require over a billion entries to be stored per matrix. For some operations it suffices to use an operation that has the same effect on the state amplitude vector (Spector [16] refers to this as "implicit matrix expansion"). For example, it is pointless to store a NOT operation in its explicit matrix expansion form. 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 these two amplitudes. Massey *et al* [21] 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 and for applying implicitly expanded gates. (Explicit matrix expansion of a gate may still be necessary, e.g. for use in forming some explicit product matrix.)

# 3 Evolved quantum artefacts

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

## 3.1 Fundamental algorithms

A number of papers authored in various combinations by Spector, Bernstein, Barnum and Swami [22, 28, 23, 24, 13] established the field of quantum genetic programming.

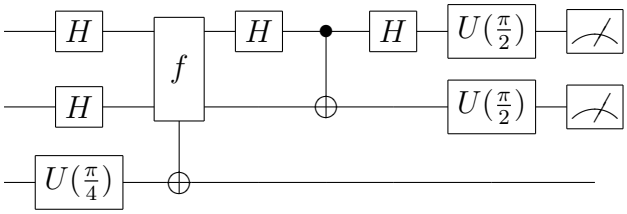



Figure 5: GP evolved (but hand simplified) Grover's four-item database search (after [16])

### 3.1.1 Deutsch-Josza promise

Some of the earliest work using GP aimed to evolve quantum circuitry to determine properties on oracle functions: given a quantum black-box function $f(q1, \ldots, qn)$ determine whether it has the property $P(f)$. The Deutsch-Josza algorithm to determine whether a function $f$ is balanced or constant (for a 1-input function $f$ this is the parity problem) is described in §B.1. [13] presents an evolved solution to the corresponding 2-bit promise problem (using traditional tree-based GP).

### 3.1.2 Grover's search

[13] also describes the evolution using SBLGP (see §2.1.2) 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 $0 \ldots 3$ and the aim is to return the single index in that range for which $f(x) = 1$, that is, there is a single 'marked' solution. See §B.2.) This is an important result, since [13] was published in 1999, a mere two years after Grover published his algorithm in 1997. Other circuits can be found in [16]. It is not uncommon for human analysts to simplify evolved artefacts (see §2.3.2). Figure 5 shows a version of Grover's solution to the four-item database: this is a hand-simplification of a GP-evolved circuit [16].

### 3.1.3 OR problem

Another simple fundamental property concern the $OR$ problem: determine whether any input $x$ gives rise to a true output $f(x)$.

[28] uses SLLGP to evolve a faster than classical solution to the OR problem. For the one qubit case an evolved circuit is shown in figure 6.

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

$$\tfrac{1}{2}\left(|0\rangle\left(|f(0)\rangle + |f(1)\rangle\right) + |1\rangle\left(|f(0)\rangle - |f(1)\rangle\right)\right)$$

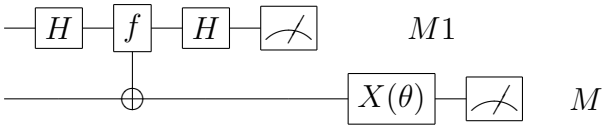



Figure 6: Circuit addressing the OR problem (after [28, fig.1])

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

$$X(\theta) = \begin{pmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{pmatrix} \tag{5}$$

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:

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

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

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

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.

### 3.1.4 AND/OR problem

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 $f(0)$, $f(1)$, ..., $f(2^n-1)$. 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

$$AND/OR_1(f) = f(0) \wedge f(1) \tag{6}$$

$$AND/OR_2(f) = (f(0) \vee f(1)) \wedge (f(2) \vee f(3)) \tag{7}$$

$$AND/OR_3(f) = ((f(0) \wedge f(1)) \vee (f(2) \wedge f(3))) \wedge ((f(4) \wedge f(5)) \vee (f(6) \wedge f(7))) \tag{8}$$

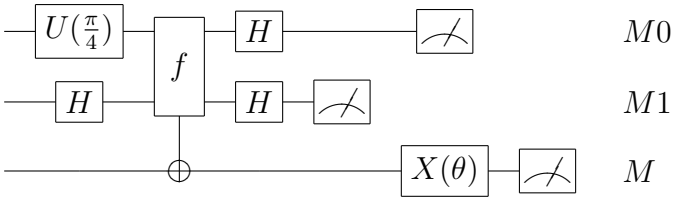



Figure 7: Faster than classical solution to 2 bit AND/OR ($\theta = 0.74909$)

| $f_{00}\ f_{01}\ f_{10}\ f_{11}$ | $p_e$ |
|---|---|
| 0000 | 0.00560 |
| 0001, 0010, 0100, 1000 | 0.28731 |
| 0011, 1100 | 0.21269 |
| 0101, 0110, 1001, 1010 | 0.28731 |
| 1101, 1110, 1011, 0111 | 0.21269 |
| 1111 | 0.00560 |

Table 1: Error probabilities for 2-bit AND/OR solution

[28] use SLLGP to evolve faster than classical solutions to the two bit AND/OR problem. Again, some hand-tuning was used to improve the evolved algorithm. The circuit diagram is shown in figure 7 with error probabilities $p_e$ for the various functions $f$ shown in table 1.

## 3.2 Hogg's algorithm

Hogg [29] demonstrates efficient quantum algorithms for attacking $k$-sat problems. Let $V_1, \ldots, V_n$ 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 $k$ $L_i$

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

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

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

Though classical algorithms for this problem are of $O(n)$, Hogg's algorithm is still more efficient. Leier & Banzhaf [15] have evolved circuits equivalent to Hogg's algorithm for the simple 1-sat case. They present some "slightly hand tuned quantum algorithms" arising from GP searches for 1-sat on 2, 3 and 4 variables (table 2).

| $n = 2$ | 3 | 4 |
|---|---|---|
| | | $H$ 0 |
| | $H$ 0 | $H$ 1 |
| $H$ 0 | $H$ 1 | $H$ 2 |
| $H$ 1 | $H$ 2 | $H$ 3 |
| $INP$ | $INP$ | $INP$ |
| $Rx[3\pi/4]$ 0 | $Rx[3\pi/4]$ 0 | $Rx[3\pi/4]$ 0 |
| $Rx[3\pi/4]$ 1 | $Rx[3\pi/4]$ 1 | $Rx[3\pi/4]$ 1 |
| | $Rx[3\pi/4]$ 2 | $Rx[3\pi/4]$ 2 |
| | | $Rx[3\pi/4]$ 3 |

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

We can readily see there is a pattern suggesting extension of the idea. Indeed the authors 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 [15]:

> 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.

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

## 3.3 Probabilistic quantum circuits

Massey *et al* [21, 30] 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 evolved program is executed to generate a quantum circuit).

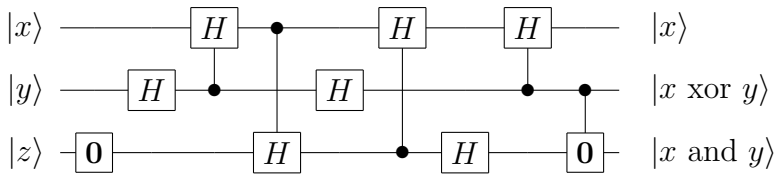



Figure 8: Probabilistic half-adder

| initial state | correct answer | probability of ending in state $\|000\rangle$ | $\|001\rangle$ | $\|010\rangle$ | $\|011\rangle$ | $\|100\rangle$ | $\|101\rangle$ | $\|110\rangle$ | $\|111\rangle$ |
|---|---|---|---|---|---|---|---|---|---|
| $\|000\rangle$ | $\|000\rangle$ | **0.53** | 0.22 | 0 | 0 | 0 | 0.09 | 0.14 | 0 |
| $\|001\rangle$ | $\|000\rangle$ | **0.53** | 0.22 | 0 | 0 | 0 | 0.09 | 0.14 | 0 |
| $\|010\rangle$ | $\|010\rangle$ | 0 | 0.05 | **0.61** | 0 | 0 | 0.09 | 0.24 | 0 |
| $\|011\rangle$ | $\|010\rangle$ | 0 | 0.05 | **0.61** | 0 | 0 | 0.09 | 0.24 | 0 |
| $\|100\rangle$ | $\|110\rangle$ | 0.09 | 0.04 | 0.03 | 0 | 0 | 0.02 | **0.82** | 0 |
| $\|101\rangle$ | $\|110\rangle$ | 0.09 | 0.04 | 0.03 | 0 | 0 | 0.02 | **0.82** | 0 |
| $\|110\rangle$ | $\|101\rangle$ | 0 | 0.30 | 0.10 | 0 | 0.02 | **0.53** | 0.04 | 0 |
| $\|111\rangle$ | $\|101\rangle$ | 0 | 0.30 | 0.10 | 0 | 0.02 | **0.53** | 0.04 | 0 |

Table 3: Probabilities of obtaining outcomes for half adder inputs

### 3.3.1 Probabilistic half adder

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 **0**. The found solution had previously been designed by Gosset [31].

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 **0**, 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 8, with probabilistic results shown in table 3.

### 3.3.2 Probabilistic max function

Q-PACE III evolved a number of probabilistic quantum programs which, when given a number of suitably encoded $[0..3] \rightarrow [0..3]$ permutation func-

| fitness case | correct answer | probability of ending in state $\|00\rangle$ | $\|01\rangle$ | $\|10\rangle$ | $\|11\rangle$ |
|---|---|---|---|---|---|
| (3,1,0,2) | $\|00\rangle, x = 0$ | **0.53** | 0.22 | 0.03 | 0.22 |
| (0,2,3,1) | $\|10\rangle, x = 2$ | 0.03 | 0.22 | **0.53** | 0.22 |
| (3,0,1,2) | $\|00\rangle, x = 0$ | **0.53** | 0.22 | 0.03 | 0.22 |
| (1,2,3,0) | $\|10\rangle, x = 2$ | 0.03 | 0.22 | **0.53** | 0.22 |
| (3,2,0,1) | $\|00\rangle, x = 0$ | **0.53** | 0.22 | 0.03 | 0.22 |
| (2,3,0,1) | $\|01\rangle, x = 1$ | 0.22 | **0.53** | 0.22 | 0.03 |
| (2,0,1,3) | $\|11\rangle, x = 3$ | 0.22 | 0.03 | 0.22 | **0.53** |
| (2,1,3,0) | $\|10\rangle, x = 2$ | 0.03 | 0.22 | **0.56** | 0.19 |

Table 4: PF MAX 1: results using 8 permutation function fitness cases

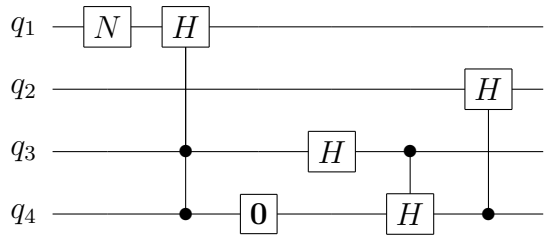



Figure 9: PF MAX 1: evolved probabilistic MAX circuit (after hand optimisation)

tions, 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.

Q-PACE III evolved the program PF MAX 1 using 8 fitness cases (expressed as permutations), with probabilistic results shown in table 4.

PF MAX 1 is a probabilistic solution to all 8 of the fitness cases used. 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 evolved circuit (after hand removal of 5 gates that have no effect) is shown in figure 9.

Repeated experiments failed to evolve a program that would give the

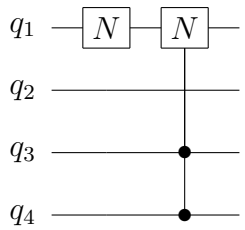



Figure 10: PF MAX 2: evolved probabilistic MAX circuit (after hand optimisation)

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 10.

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 $|x11\rangle$ 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 (although it is acknowledged that the degree of improvement supplied by this generalisation decreases exponentially as $n$ increases.)

## 3.4 Quantum Fourier Transform

Massey *et al* [32, 30] report the results of using two quantum genetic programming suites, QPACE-III and QPACE-IV, to evolve a variety of QFT

circuits (see §B.3). QPACE-III uses a second order encoding (where the evolved program is executed to generate a quantum circuit); QPACE-IV additionally includes parameters, so that the precise circuit generated can be a function of, for example, the number of qubits.

QPACE-III successfully evolved a 3-qubit QFT. The solution, like many evolved solutions, is irregular, with little discernable structure.

When QPACE-IV was used on test cases of 1, 2, and 3 qubits, it successfully evolved a parameterised circuit that implements the QFT for also 1, 2, and 3 qubits, but not for higher numbers of qubits. When QPACE-IV was used on test cases of 1, 2, 3, and 4 qubits, it successfully evolved a parameterised circuit that implements the QFT for *all* numbers of qubits: there is enough information in the extra test cases to allow it to generalise correctly.

Evolving such a parameterised solution necessarily results in circuits with more regular structures.

## 3.5 Efficient implementation

### 3.5.1 Introduction

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 has been 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* [33] point out, there is "enormous potential for simplifying the implementation of working quantum computers". This section reviews evoutionary search for *efficient* implementations of quantum artefacts.

### 3.5.2 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 implemented (although the general motivation is clearly sensible, this requirement may be challenged: Williams & Gray [12] note "An incomplete gate set may make sense when the properties of the target computation allow it.") 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.

### 3.5.3 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 [34] 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 Gershenfeld and Chuang's $CN$ gate implementation is (up to phase, which can be removed by further rotations)

$$CN_{12} = R_{y1}(-90)R_{z2}(-90)R_{z1}(-90)R_{z12}(180)R_{y1}(90) \tag{9}$$

There is a choice of pulse sequences (each implementing a rotation) to implement $CN$. Where there is choice, there is potential optimisation. Rethi-

nam *et al* [33] 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 of rotation, allowing an accuracy one degree. They evolved rotation sequences of length 3, more efficient than previously exhibited solutions. These are

$$CN_{12} = R_{z2}(270)R_{xz12}(90)R_{x1}(90) \quad (10)$$

$$CN_{12} = R_{x1}(90)R_{z2}(270)R_{xz12}(90) \quad (11)$$

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 $\theta$ as a parameter. Small changes in $\theta$ 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 [33] is an important contribution.

### 3.5.4 Higher level basic gates

Lukac *et al* [19] 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.

Khan & Perkowski [35] focus on *qutrits*: three-valued quantum variables. They evolve a variety of ternary quantum circuits, with a strong emphasis on efficiency. Their fitness function includes various terms to minimise the resources used, in particular, minimising the *scratchpad resources*, or number of wires in the circuit. They have found more efficient realisations of certain basic ternary gates.

### 3.5.5 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 take place only 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 [36]. There are also choices to be made as to the physical location where gates will be applied. Van Meter & Binkley [36] 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.

### 3.5.6 Optimising existing solutions

As noted earlier, given a fully working circuit, you may wish to optimise some of its non-functional properties. A common approach to such problems in computer science is to apply a succession of *semantics-preserving transformations*, 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 side with the right hand side might give different results in an implementation using floating point numbers.)

Maslov *et al* [26] 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 quantum circuit optimisation

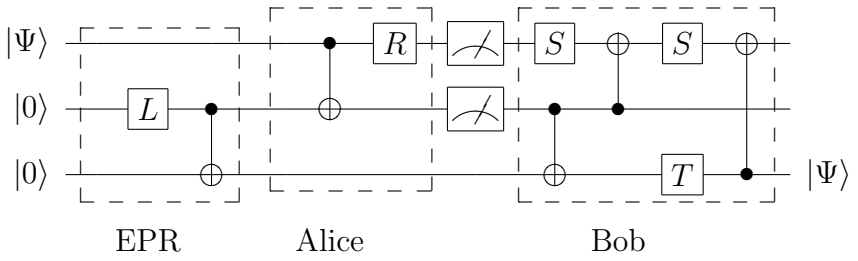



Figure 11: Teleportation circuit of Brassard [38]

problem.

## 3.6 Communication, teleportation, entanglement

We can be fairly flexible as to what counts as an algorithm or program. As Clark & Jacob [37] 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.

### 3.6.1 Quantum teleportation

Several evolutionary search researchers have attempted to evolve circuits for *quantum teleportation*: 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 [38] is shown in figure 11, where the single qubit gates are defined by:

$$L = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}; \qquad R = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \tag{12}$$

$$S = \begin{pmatrix} i & 0 \\ 0 & 1 \end{pmatrix}; \qquad T = \begin{pmatrix} -1 & 0 \\ 0 & -i \end{pmatrix} \tag{13}$$

As is usual in protocols work, communication is between Alice and Bob. (Alice, Bob, and Eve, who eavesdrops on communications between Alice and Bob, are the traditional actors in descriptions of classical secure communications protocols.)

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

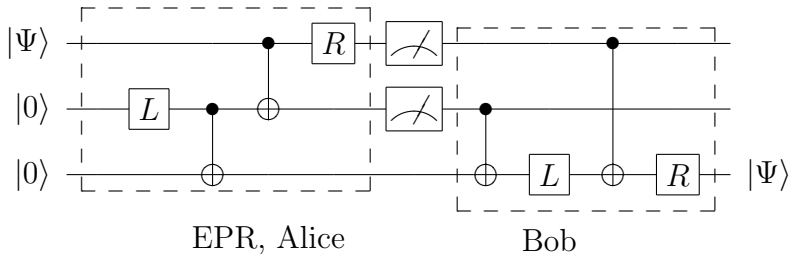



Figure 12: Evolved teleportation circuit of Williams & Gray [12]

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 [12] 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, which is able to produce a variety of send circuits of similar efficiency to the Brassard circuit and improved receive circuits. Furthermore, the system allows the user to restrict the choice of gates. A complete evolved circuit using only $L$, $R$ and $CN$ is shown in figure 12.

Williams & Gray [12] use a cost function that measures deviation of the evolved unitary matrix $S$ from the (assumed known) target matrix $U$ (see §2.2.1).

Yabuki & Iba [20] 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 on only 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 form $\{0, 1, 2, 3\}$. 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 to start 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 overall interpretation is based on a different table for each section, see table 5. (For consistency within this review, we have adopted the convention of $C_{ij}$ denoting control qubit $i$ and target qubit $j$. Yabuki & Iba [20] reverse this convention.)

A chromosome in this scheme that describes William & Gray's circuit [12] (figure 12) is shown in figure 13. The first codon 112 indicates the element in the major row 1, column indexed 1, minor row 2 of the EPR table, hence

| | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | $CN_{10}$ | $CN_{01}$ | | | | $CN_{21}$ | $CN_{12}$ | | | $CN_{01}$ | $CN_{02}$ | | 0 |
| | $CN_{10}$ | $CN_{01}$ | | | | $CN_{21}$ | $CN_{12}$ | | $CN_{10}$ | | $CN_{12}$ | | 1 |
| | $CN_{10}$ | $CN_{01}$ | | | | $CN_{21}$ | $CN_{12}$ | | $CN_{20}$ | $CN_{21}$ | | | 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** | | | | |

Table 5: Codon interpretation tables for three stages

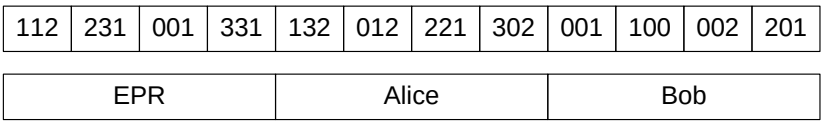



Figure 13: Chromosome corresponding to teleportation circuit [12]

$L_1$. The second codon maps to an empty element, and so is ignored. 001 decodes to $CN_{10}$. 331 marks the start of Alice's part. 132 is ignored; 012 is $CN_{21}$; 221 is $R_2$. 302 is interpreted as Alice's measurement. Bob's four codons are $CN_{10}$, $L_0$, $CN_{20}$ and $R_0$.

Yabuki & Iba [20] evolved a simpler circuit, shown in figure 14. 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, that is, how well it transfers the source qubit state exactly.

The reader may well be struck by just how *small* 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 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.

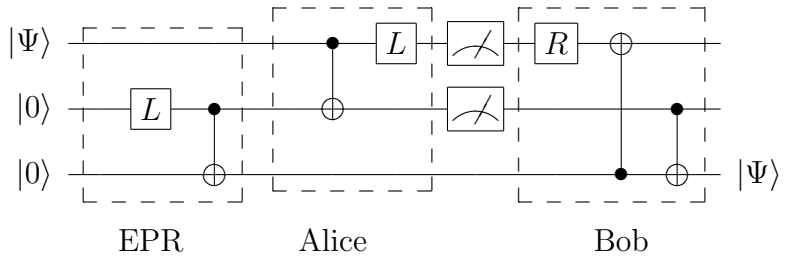



Figure 14: Yabuki & Iba's improved teleportation circuit [20]

### 3.6.2 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 [8]). The tradeoffs between classical communication and quantum entanglement resources are not yet well 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 [39] report Smolin as suggesting the gate in equation 14) as being capable of generating entanglement but not classical communication.

$$SMOLIN = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{1}{\sqrt{2}} & 0 & 0 & -\frac{1}{\sqrt{2}} \end{pmatrix} \tag{14}$$

Spector & Bernstein [39] 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 15, where

$$J(\theta) = \begin{pmatrix} \cos\theta & 0 & 0 & \sin\theta \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \sin\theta & 0 & 0 & -\cos\theta \end{pmatrix} \tag{15}$$

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.

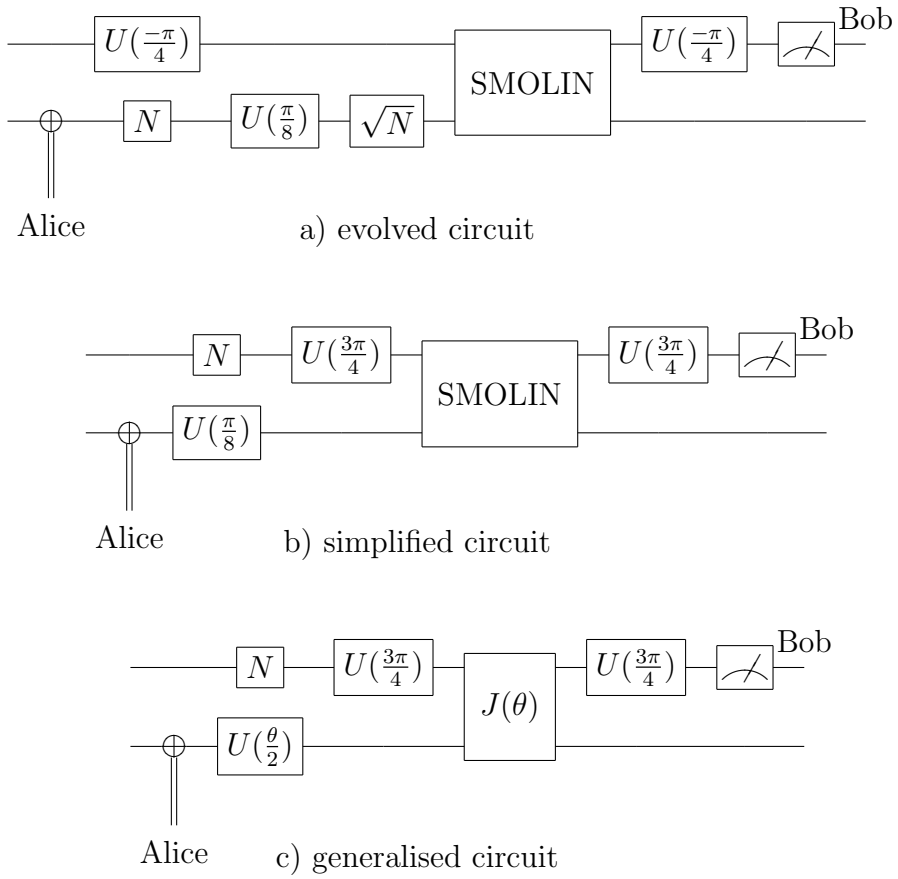



Figure 15: From evolved circuit to general idea

### 3.6.3 Entanglement

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 [40] 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 in figure 15, patterns can be recognised.)

It is not always necessary to use the full sophistication of evolutionary algorithms: Brown *et al* [41] successfully searched for highly entangled states using a simple hill-climbing algorithm.

Spector & Bernstein [39] demonstrate an evolved circuit that allows two bits of classical information to be communicated with one-bit of prior entanglement, as shown in Figure 16.

We believe that the exploration of entanglement and communication

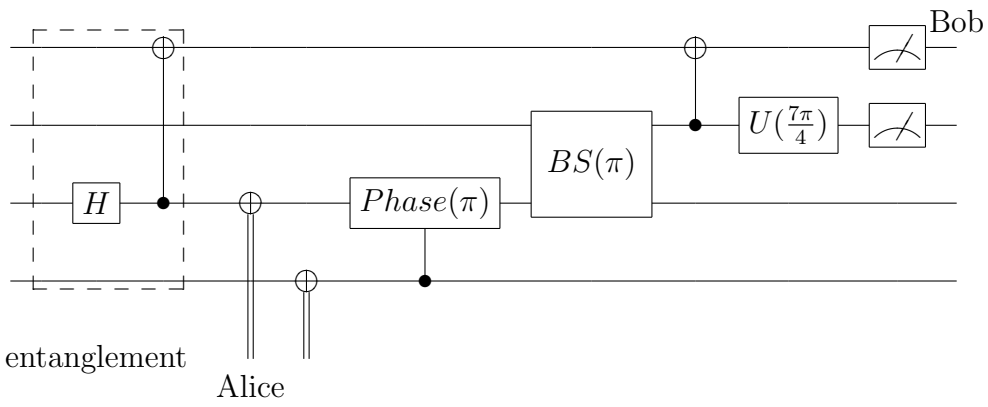



Figure 16: One-bit prior entanglement allows two classical bits communication [39]

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.

## 3.7 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 se that pieces of work are underway on fundamental problems of the topic.

# 4 The Future

This section is rather more speculative. First we draw some conclusions from the results so far achieved. Then, based on theses, 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.

## 4.1 Conclusions: the story so far

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. 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. 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 suc-

cess 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.

## 4.2 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 [42].

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 [43], and Massey *et al*'s 'row swapping' optimisation [21]. 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 [44]. The quantum circuit search space appears to be exceptionally rugged [27], 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.

## 4.3 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, [8, 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 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.

Spector [16, 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 17.

Spector shows the progress of Grover's algorithm on 3 qubits as a sequence of cube diagrams [16, 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 18. This suggests that a pair of hypercubes (or a

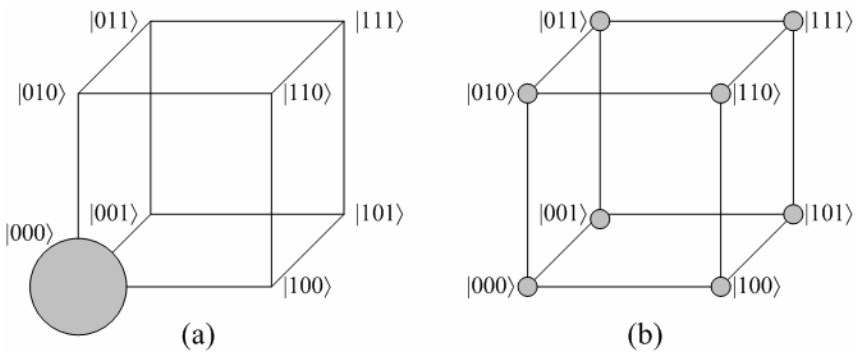



Figure 17: A 3-qubit Spector cube for (a) the state $|000\rangle$ (b) equal superposition of all states

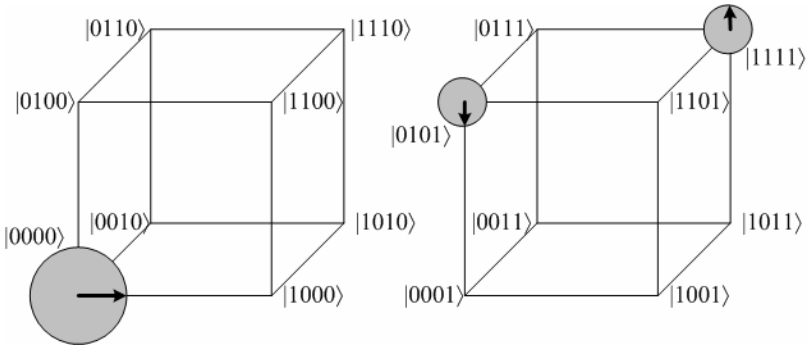



Figure 18: A 4-qubit complex Spector cube for the state $\sqrt{\frac{1}{6}}(2|0000\rangle - i|0101\rangle + i|1111\rangle$

pair of pairs of cubes) could be used to represent a 5-qubit state.

## 4.4 Improved search on rugged landscapes

Quantum search landscapes offer a particular challenge to evolutionary algorithms, since they appear to be exceptionally rugged [27]. Ruggedness is a general problem in search, and there seems to be no reason to expect *a priori* that evolution, good at searching biological spaces, should be *naively* applicable to other, artificial, search spaces. Current evolutionary algorithms are certainly nave when compared to the vast richness of the biological processes [45].

We believe a more advanced approach is appropriate, examining more sophisticated biological understandings of evolutionary search, and more sophisticated design of the representations and search spaces.

### 4.4.1 Representation

The choice of representation of the problem domain affects the ruggedness of the landscape, and the comprehensibility of the solutions. Even a trivial change in representation, say to Gray coding, affects the shape of the search space tremendously. Other changes, such as embedding in higher dimensions, can have enormous effects on search efficiency (see, for example, [46]).

The languages used to represent the problem can also effect the understandability, and generality, of the solution. For example, using higher level search languages (encoding algorithms, rather than quantum circuits) leads to more regular, and hence more comprehensible, solutions (§3.4).

We believe that the effect of quantum problem representation on search efficiency should be explored, particularly the effect of embedding in higher dimensions and projecting into lower dimensions, and the use of various higher level representation language designs.

### 4.4.2 Trajectories

Traditional evolutionary search is amnesiac: the next search step depends only on the current population, not on how it was achieved. Yet there is an enormous amount of valuable information available in the search *trajectory*: the history of populations past (see, for example, [47]).

We believe that if the search trajectory through rugged quantum landscapes were analysed, it could used to achieve significantly better results than those found by the final population alone. In particular, if the final population fails to capture a solution, analysis of trajectory information may nevertheless yield a (partial) solution.

### 4.4.3 Predator-prey coevolution

Lewis Carroll's Red Queen told Alice that "it takes all the running you can do to keep in the same place" [48]. Van Valen [49] argues that biological evolution is like this because the environment of one species is constantly made more difficult by the evolution of other species.

One way to implement the concept of coevolution in evolutionary computing is to evolve the problems to become harder as the solutions become better [50]; a population of problems is itself subject to evolution in which fitness is inversely related to the ease with which the current population of algorithms can solve them. In this way, the initial solution of relatively easy problems can lead continuously to more powerful algorithms, without subjecting the early attempts to the full difficulty (ruggedness) of the final required solution.

### 4.4.4 Higher-level approaches

Second-order encodings (see §2.1.3) seem to support for more structured, scalable solutions. Such higher-level approaches could be exploited more. One such approach is Ryan & O'Neill's *Grammatical Evolution* [51, 52], 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.

## 4.5 New quantum problems to explore

The majority of the evolved artefacts so far discovered replicate known results. This is unsurprising: it is relatively early days, and it is safest to search in places where one knows there *are* results, and knows that one will recognise them when one finds them. However, having now proved the concept that non-trivial quantum computing artefacts can be discovered by evolutionary search, it is time to move beyond reproducing known results into more challenging areas. In this section we suggest some areas that might be fruitful.

### 4.5.1 Quantum error correction

Bell inequalities [53] demonstrate the existence of quantum mechanical (or, rather, non-classical) effects. However, their physical implementation and experimental evaluation relies on certain data sampling assumptions in the presence of detector inefficiencies. It is useful to attempt to devise forms of these inequalities that are robust to such problems. The brute force search approach is reaching at the limit of conventional computational power, with each search for a 12-particle inequality requiring roughly 6-CPU months of search time[1]. One could next move to a supercomputer, but we suggest that an evolutionary search technique would also be applicable to this search problem.

Linear-optics quantum error correction circuits [54] would allow for easy integration into a quantum technology infrastructure, by moving the difficulties into resource production. In addition, these circuits have a classical limit that performs useful error correction [55]. The search space, although looking at 'small' problems, is still huge, because the variables are continuous. There is currently known a 9-wavepacket code [54] (an analogue of Shor's original

[1] Sam Braunstein, private communication

9-qubit code), but this would be exceedingly difficult to implement experimentally. Instead, it is hoped that a linear-optics analogue of the known 5-qubit code could be constructed, which has a much better chance of being constructed in the lab. We suggest that an evolutionary search technique might be able to discover such a 5-wavepacket code.

### 4.5.2 Searching for extreme examples

Meta-heuristic search has been successfully used to find counter-examples to conjectures in classical domains, such as classical cryptography [56, 18]. There are many conjectures in quantum computation and quantum information theory (such as bounds on entanglement, and channel capacities); These offer suitable targets to meta-heuristic searches for counterexamples. For example, Spector & Bernstein [39] use genetic programming to discover new bounds on quantum communication. More sophisticated cases are yet to be tackled by search, for example conjectures on the number of mutually unbiased basis vectors in non-prime power dimensions [57].

### 4.5.3 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 [58]) 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.

### 4.5.4 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 [37], using the classical protocol reasoning BAN logic [59]. Can this approach be extended to quantum protocols? How do we need to extend or change the logics to handle quantum protocols?

### 4.5.5 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 (see §3.5.4 for a qutrit example). It is not even necessary to restrict the quantum values to be discrete: continuous quantum algorithms exist [60], and might form a 'smoother' search space. (Of course, the search space is also larger, as noted in §4.5.1.)

Non-circuit based models of quantum computation, such as quantum cellular automata [61, 62], and measurement-based quantum machines [63], could prove to have more tractable search spaces.

## 4.6 Quantum search

One reason for interest in quantum algorithms is their increased efficiency. We are interested in metaheuristic search as an efficient (if approximate) way of exploring the huge search spaces inhabited by quantum artefacts. Can we come full circle, and use an efficient *quantum* search algorithm to search for quantum artefacts?

Grover's algorithm uses quantum effects to perform searches more efficiently that can classical algorithms, although it is only a square root, not an exponential, speedup over the classical case. Clark and Stepney [64] suggest a hybrid approach to search (in a cryptographic context): use a classical search to reduce a huge search space to a size that makes Grover's algorithm feasible.

### 4.6.1 Quantum random walks

Quantum random walks [65] 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* [66] 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 [67].

### 4.6.2 Quantum genetic algorithms

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 [68]. Can these two global, parallel processes be combined to produce a quantum genetic algorithm?

Various authors have claimed quantum-inspired genetic algorithms, inspired by the concept of interference [69] or superposition of single qubits [70, 71]. However, these algorithms use representations that are entirely classical, in that they do not support *entanglement* of multiple qubits. Thus they fail to access the exponential state space size (and hence computational power) of an entangled quantum system. Additionally, the quantum genetic operators introduced are not unitary. Hence these cannot be considered to be true quantum genetic algorithms.

More detailed work by Rylander *et al* [72] suggests that getting a fruitful combination is non-trivial, as the genetic operators such as crossover are tricky to move to the quantum domain.

Udrescu *et al* [73] point out that a single quantum (binary) chromosome can hold a superposition of all possible states, and a corresponding fitness register can hold the corresponding fitnesses. Given an oracle that can mark the fittest states (all the states greater than some $f_{max}$, say), then Grover's search can be used to find these states, and no quantum genetic operators are required. However, this approach is infeasible for large chromosomes, as this final search would still take time exponential in the length of the chromosome, $n$, growing as $O(\sqrt{2^n})$. Note that Grover's search is applicable to unstructured data, whereas a genetic algorithm attempts to exploit structure in the search space, in order to reduce an infeasible problem to a feasible one.

So it seems we will need yet further intuition priming to understand how to combine quantum inspiration with classical results.

# References


[1] Susan Stepney and John A. Clark. Evolving quantum programs and protocols. In Michael Rieth and Wolfram Schommers, editors, *Handbook of Theoretical and Computational Nanotechnology, volume 3, Quantum and Molecular Computing, Quantum Simulations*, chapter 3, pages 113–160. American Scientific Publishers, 2006.

[2] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, Sept 1991.

[3] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 1965.

[4] Richard P. Feynman. Simulating physics with computers. *Int. J. Theor. Phys.*, 21(6/7):467–488, 1982.

[5] 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.

[6] Paul Benioff. Quantum mechanical hamiltonian models of turing machines that dissipate no energy. *Phys. Rev. Lett.*, 48:1581–1585, 1982.

[7] David Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. *Proc. R. Soc. Lond. A*, 400:97–117, 1985.

[8] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[9] Peter Shor. Algorithms for quantum computation: discrete log and factoring. In *Proc. 35th Ann. Symp. Foundations of Computer Science*, pages 124–134. IEEE, 1994.

[10] Dirk Boumeister, Artur Ekert, and Anton Zeilinger, editors. *The Physics of Quantum Information: Quantum Cryptography, Quantum Teleportation, Quantum Computation*. Springer, 2000.

[11] Colin P. Williams and Scott H. Clearwater. *Explorations in Quantum Computing*. Springer, 1997.

[12] Colin P. Williams and Alexander G. Gray. Automated design of quantum circuits. In *Quantum Computing and Quantum Communications: First NASA Conference (QCQC'98)*, LNCS 1509, pages 113–125. Springer, 1999.

[13] Lee Spector, Howard Barnum, Herbert J. Bernstein, and Nikhil Swamy. Quantum computing applications of genetic programming. In *Advances in Genetic Programming 3*, pages 135–160. MIT Press, 1999.

[14] Wolfgang Kantschik and Wolfgang Banzhaf. Linear tree GP and its comparison with other GP structures. In *EuroGP 2001*, LNCS 2038, pages 302–312. Springer, 2001.

[15] André Leier and Wolfgang Banzhaf. Evolving Hogg's quantum algorithm using linear-tree GP. In *GECCO 2003* [74], pages 390–400.

[16] Lee Spector. *Automatic Quantum Computer Programming: a genetic programming approach*. Kluwer, 2004.

[17] D. DiVincenzo and J. Smolin. Results on two-bit gate design on quantum computers. In W. Porod and eds G. Frazier, editors, *Proc. 2nd Workshop on Physics and Computation (PhysComp '94)*, pages 14–23. IEEE, 1994.

[18] John A. Clark, Jeremy L. Jacob, and Susan Stepney. Searching for cost functions. In *CEC 2004* [75], pages 1517–1524.

[19] Martin Lukac, Marek Perkowski, Hilton Goi, Mkhail Pivtoraiko, Chung Hyo Yu, Kyusik Chung, Hyunkoo Jee, Byung guk Kim, and Yong-Duk Kim. Evoluationary approach to quantum reversible circuits synthesis. *Artificial Intelligence Review*, 20:361–417, 2003.

[20] Taro Yabuki and Hotoshi Iba. Genetic algorithms for quantum circuit design – evolving a simpler teleportation circuit. In *Late Breaking Papers at GECCO 2000*, pages 425–430, August 2000.

[21] Paul Massey, John A. Clark, and Susan Stepney. Evolving quantum circuits and programs through genetic programming. In *GECCO 2004* [76], pages 569–580.

[22] Howard Barnum, Herbert J. Bernstein, and Lee Spector. A quantum circuit for or. quant-ph/990756, 1999.

[23] L. Spector, H. Barnum, H. J. Bernstein, and N. Swamy. Genetic programming for quantum computers. In *Genetic Programming 1998*, pages 365–374. Morgan Kauffman, 1998.

[24] L. Spector, H. Barnum, H. J. Bernstein, and N. Swamy. Finding a better-than-classical quantum and/or algorithm using genetic programming. In *CEC 1999*, pages 2239–2246. IEEE, 1999.

[25] Chris Lomont. Quantum circuit identities. quant-ph/0307111, 2003.

[26] D. Maslov, C. Young, D. M. Miller, and G. W. Dueck. Quantum circuit simplification using templates. In *Proc. Conference on Design, Automation and Test Europe (DATE 05)*, pages 1208–1213. IEEE, 2005.

[27] André Leier and Wolfgang Banzhaf. Exploring the search space of quantum programs. In *CEC 2003*, pages 170–177. IEEE Press, 2003.

[28] H. Barnum, H. J. Bernstein, and L. Spector. Quantum circuits for or and and of ors. *J. Physics A: Mathematical and General*, 33(45):8047–8057, November 2000.

[29] Tad Hogg. Solving highly constrained search problems with quantum computers. *J. Artificial Intelligence Research*, 10:39–66, 1999.

[30] Paul Massey, John A. Clark, and Susan Stepney. Human-competitive evolution of quantum computing artefacts by genetic programming. *Evolutionary Computation Journal*, 14(1):22–40, 2006.

[31] Phil Gossett. Quantum carry-save arithmetic. quant-ph/9808061, 1998.

[32] Paul Massey, John A. Clark, and Susan Stepney. Evolution of a human-competitive Quantum Fourier Transform algorithm using genetic programming. In *GECCO 2005*, pages 1657–1664. ACM Press, 2005.

[33] Manoj Jesu Rethinam, Amil Kumar Javali, E. C. Behrman, J. E. Steck, and S. R. Skinner. A genetic algorithm for finding pulse sequences for NMR quantum computing. quant-ph/0404170 v1, April 2004.

[34] Neil A. Gershenfeld and Isaac L. Chuang. Bulk spin-resonance quantum computing. *Science*, 275:350–356, January 1997.

[35] M. H. A. Khan and M. Perkowski. Genetic algorithm based synthesis of multi-output ternary functions using quantum cascade of generalized ternary gates. In *CEC 2004* [75], pages 2194–2201.

[36] Rodney Van Meter and Kevin Binkley. Compiling quantum programs using genetic algorithms. In *The Wild and Crazy Idea Session IV and abstracts and part of 11th Intl. Conf. Architectural Support for Programming Languages and Operating Systems*, October 2004.

[37] John A. Clark and Jeremy L. Jacob. Protocols are programs too: the meta-heuristic search for security protocols. *Information & Software Technology*, 43(14):891–904, December 2001.

[38] G. Brassard. Teleportation as quantum computation. In *Proc. 4th Workshop on Physics and Computation, New England Complex Systems Institute 1996*, 1996. Also as quant-ph/9605035.

[39] L. Spector and H. J. Bernstein. Communication capacities of some quantum gates and discovered in part through genetic programming. In *Proc. 6th Int. Conf. Quantum Communication and Measurement and and Computing (QCMC)*, pages 500–503. Rinton Press, 2003.

[40] Ben I. P. Rubinstein. Evolving quantum circuits using genetic programming. In *CEC 2001* [77], pages 144–151.

[41] Iain Brown, Susan Stepney, Anthony Sudbery, and Samuel L. Braunstein. Searching for multi-qubit entanglement. *Journal of Physics A*, 38:1119–1131, 2005.

[42] André Leier and Wolfgang Banzhaf. Comparison of selection strategies for evolutionary quantum circuit design. In *GECCO 2004* [76], pages 557–568.

[43] George F. Viamontes, Manoj Rajagopalan, Igor L. Markov, and John P. Hayes. Gate-level simulation of quantum circuits. quant-ph/0208003, 2002.

[44] Andrew J. Booker, J. E. Dennis Jr., Paul D. Frank, David B. Serafini, Virginia Torczon, and Michael W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural Optimization*, 17(1):1–13, 1999.

[45] Susan Stepney, Tim Clarke, and Peter Young. Plazzmid: An evolutionary agent-based architecture inspired by bacteria and bees. In *ECAL 2007, Lisbon, Portugal, September 2007*, LNCS. Springer, 2007.

[46] John A. Clark, Jeremy L. Jacob, S. Maitra, and P. Stanica. Almost Boolean Functions: the design of boolean functions by spectral inversion. *Computational Intelligence*, 20(3):450–462, 2004.

[47] John A. Clark and Jeremy L. Jacob. Fault injection and a timing channel on an analysis technique. In *Eurocrypt 2002*, LNCS 2332, pages 181–196. Springer, 2002.

[48] Lewis Carroll. *Through the Looking-Glass (and what Alice found there).* Macmillan & Co, 1872.

[49] Leigh Van Valen. A new evolutionary law. *Evolutionary Theory*, 1:1–30, 1973.

[50] W. Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D*, 42:228–234, 1990.

[51] Michael O'Neill and Conor Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language.* Kluwer, 2003.

[52] Conor Ryan, J. J. Collins, and Michael O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *EuroGP'98*, LNCS 1391, pages 83–96. Springer, 1998.

[53] John S. Bell. On the Einstein-Podolsky-Rosen paradox. *Physics*, 1:195–200, 1964.

[54] Samuel L. Braunstein. Error correction for continuous quantum variables. *Phys. Rev. Lett.*, 80:4084–4087, 1998.

[55] Samuel L. Braunstein. Quantum error correction for communication with linear optics. *Nature*, 394:47–49, 1998.

[56] John A. Clark, Jeremy L. Jacob, Susan Stepney, S. Maitra, and William Millan. Evolving boolean functions satisfying multiple criteria. In *INDOCRYPT 2002*, LNCS 2551, pages 246–259. Springer, 2002.

[57] M.. Planat, H. C. Rosu, and S. Perrine. A survey of finite algebraic geometrical structures underlying mutually unbiased quantum measurements. *Foundations of Physics*, 36(11):1662–1680, 2006.

[58] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[59] Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. Technical Report SRC Research report 39, February 1989. http://gatekeeper.research.compaq.com/pub/DEC/SRC/research-reports/abstracts/src-rr-039.html.

[60] S. L. Braunstein and A. K. Pati, editors. *Quantum Information with Continuous Variables*. Springer, 2003.

[61] Gavin K. Brennan and Jamie E. Williams. Entanglement dynamics in one-dimensional quantum cellular automata. *Phys. Rev. A*, 68:042311, 2003.

[62] B. Schumacher and R.F. Werner. Reversible quantum cellular automata. quant-ph/0405174, 2004.

[63] Simon Perdrix and Philippe Jorrand. Measurement-based quantum Turing Machines and their universality. quant-ph/0404146, 2004.

[64] John A. Clark and Susan Stepney. Fusing natural computational paradigms for cryptanalysis. or, using heuristic search to bring cryptanalysis problems within quantum computational range. In *CEC 2006*, pages 200–206. IEEE, 2006.

[65] Julia Kempe. Quantum random walks – an introductory overview. *Contemporary Physics*, 44(4):307–327, 2003.

[66] Andrew M. Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A. Spielman. Exponential algorithmic speedup by quantum walk. In *Proc. 35th ACM Symposium on Theory of Computing (STOC 2003)*, pages 59–68. ACM, 2003.

[67] Neil Shenvi, Julia Kempe, and K. Birgitta Whaley. A quantum random walk search algorithm. *Phys. Rev. A*, 67(5):052307, 2003.

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

[69] Ajit Narayanan and Mark Moore. Quantum inspired genetic algorithms. In *International Conference on Evolutionary Computation, ICEC-96*, pages 61–66. IEEE, 1996.

[70] Kuk-Hyun Han and Jong-Hwan Kim. Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *IEEE Trans. Evol. Comp.*, 6(6):580–593, December 2002.

[71] Kuk-Hyun Han and Jong-Hwan Kim. Quantum-inspired evolutionary algorithms with a new termination criterion, $h_\varepsilon$ gate, and two-phase scheme. *IEEE Trans. Evol. Comp.*, 8(2):156–169, April 2004.

[72] Bart Rylander, Terry Soule, James Foster, and Jim Alves-Foss. Quantum evolutionary programming. In *GECCO 2001*, pages 1005–1011. Morgan Kauffman, 2001.

[73] Mihai Udrescu, Lucian Prodan, and Mircea Vladutiu. Implementing quantum genetic algorithms: a solution based on Grover's algorithm. In *Proc. 3rd Conference on Computing Frontiers, Ischia, Italy, May 2006*, pages 71–82. ACM, 2006.

[74] *Genetic and Evolutionary Computation Conference: GECCO 2003, Chicago IL, USA, July 2003*, LNCS 2724. Springer, 2003.

[75] *International Conference on Evolutionary Computation: CEC 2004, Portland OR, USA, June 2004*. IEEE, 2004.

[76] *Genetic and Evolutionary Computation Conference: GECCO 2004, Seattle, USA, June 2004*, LNCS 3103. Springer, 2004.

[77] *International Conference on Evolutionary Computation: CEC 2001, Seoul, South Korea, May 2001*. IEEE Press, 2001.

[78] Eleanor G. Rieffel and Wolfgang Polak. An introduction to quantum computing for non-physicists. *ACM Computing Surveys*, 32(3):300–335, September 2000.

[79] David Deutsch and Richard Josza. Rapid solution of problems by quantum computation. *Proc. R. Soc. Lond. A*, 439:553–558, 1992.

[80] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proc. 28th Ann. ACM Symp. on the Theory of Computing (STOC)*, pages 212–219, 1996.

[81] Hoi-Kwok Lo, Sandu Popescu, and Tim Spiller, editors. *Introduction to Quantum Computation and Information*. World Scientific Publishing, 1998.

[82] N. David Mermin. From classical state-swapping to quantum teleportation. quant-ph/0105177 v4, 2002.

[83] Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2000.

[84] Sanjeev Kumar and Peter J. Bentley, editors. *On Growth, Form and Computers*. Elsevier, 2003.

[85] Leandro de Castro and Jonathan Timmis. *Artificial Immune Systems: a new computational intelligence approach*. Springer, 2002.

[86] Ingo Rechenberg. *Evolutionsstrategie: Optimierung Technisher Systeme nach Prinzipien der Biologischen Evolution*. Frommann-Holzboog, 1963.

[87] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, 1996.

[88] Lawrence J. Fogel. *Biotechnology: Concepts and Applications*. Prentice Hall, 1963.

[89] John H. Holland. Genetic algorithms and the optimum allocation of trials. *SIAM J. Comp.*, 2(2):88–105, 1973.

[90] David E. Goldberg. *Genetic Algorithms in Search and Optimization and Machine Learning*. Addison-Wesley, 1989.

[91] David E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms.* Kluwer, 2002.

[92] Melanie Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, 1996.

[93] G. R. Harik, F. G. Lobo, and David E. Goldberg. The compact genetic algorithm. *IEEE Trans. Evol. Comp.*, 3(4):287–297, 1999.

[94] Allen E. Nix and Michael D. Vose. Modeling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence*, 5(1):79–88, 1992.

[95] John R. Koza. *Genetic Programming: on the programming of computers by means of natural selection.* MIT Press, 1992.

[96] John R. Koza. *Genetic Programming II: automatic discovery of reusable programs.* MIT Press, 1994.

[97] John R. Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane. *Genetic Programming III: Darwinian invention and problem solving.* Morgan Kaufmann, 1999.

[98] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic Programming IV: routine human-competitive machine intelligence.* Kluwer, 2003.

[99] Kenneth E. Kinnear Jr, editor. *Advances in Genetic Programming.* MIT Press, 1994.

[100] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming, An Introduction: on the automatic evolution of computer programs and its applications.* Morgan Kaufmann, 1998.

[101] Guy L. Steele. *Common Lisp the Language and 2nd edition.* Digital Press, 1990.

# A Quantum circuits

This appendix gives a brief overview of the quantum concepts necessary for the paper: qubits, Dirac notation, unitary operations, and the pictorial representation of quantum gates and circuits. A good introduction to these

concepts can be found in [8, 78]. This appendix assumes an understanding of complex numbers and matrix operations.

A classical computational bit can be in one of two states: 0 or 1. A corresponding two-state quantum bit, or *qubit*, may similarly be in one of two computational 'basis states', denoted by $|0\rangle$ and $|1\rangle$, but can also exist in a complex *superposition* of these states. A superposition $|\Psi\rangle$ is denoted by $|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ where the coefficients $\alpha$ and $\beta$ are complex numbers normalised so that $|\alpha|^2 + |\beta|^2 = 1$.

The state is not directly observable as a superposition. When observed, the state is found to be $|0\rangle$ with probability $|\alpha|^2$, or $|1\rangle$ with probability $|\beta|^2$. $\alpha$ and $\beta$ are complex *probability amplitudes*.

The notation $|\Psi\rangle$ is the conventional *Dirac notation* shorthand for a column vector:

$$|0\rangle \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \quad |1\rangle \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{16}$$

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \equiv \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{17}$$

So a one-qubit system forms a 2 dimensional space, spanned by the 2 orthonormal basis vectors $|0\rangle$ and $|1\rangle$.

A quantum *operation* is a reversible operation, represented as a unitary matrix acting on the relevant state vector. (A matrix $U$ is *unitary* iff $UU^\dagger = U^\dagger U = I_n$, where $U^\dagger$ is the complex conjugate transpose of $U$, $U^\dagger = U^{*T}$, and $I_n$ is the $n \times n$ identity matrix.) For example, the unitary NOT operation $N$ is

$$N \equiv \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix};\ N\,|\Psi\rangle \equiv \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix} \equiv \beta\,|0\rangle + \alpha\,|1\rangle \tag{18}$$

One important quantum gate is the Hadamard gate, which turns pure states into superpositions:

$$H \equiv \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix};\ H\,|0\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle);\ H\,|1\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{19}$$

Multiple quantum operations are combined by matrix multiplication (it is easy to see that the product of two unitary matrices is also unitary).

An $n$ qubit system forms a $2^n$ dimensional space, spanned by $2^n$ orthonormal basis vectors $|k\rangle$ (a particular $k$ is conventionally written in binary notation, where each of the $n$ individual digits correspond to one of the $n$ qubits). Any state vector $|\Psi\rangle$ may be written in terms of its components with respect to this basis, $|\Psi\rangle = \sum_{k=0}^{2^n-1} \alpha_k\,|k\rangle$, where the $\alpha_k$ are the normalised complex

probability amplitudes, $\sum_{k=0}^{2^n-1} |\alpha_k|^2 = 1$. So, for example, a two qubit state vector can be written

$$|\Phi\rangle = \alpha_0 |00\rangle + \alpha_1 |01\rangle + \alpha_2 |10\rangle + \alpha_3 |11\rangle \equiv \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} \tag{20}$$

A unitary operation on an $n$ qubit system is represented by a $2^n \times 2^n$ unitary matrix. For example, consider the 2-qubit *controlled not*, or $CN$, operator that flips the value of the second qubit if the first has a value of 1, and leaves it unchanged if the first has a value of 0.

$$CN = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{21}$$

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

$$\alpha_0 |00\rangle + \alpha_1 |01\rangle + \alpha_2 |10\rangle + \alpha_3 |11\rangle \xrightarrow{NOT} \alpha_2 |00\rangle + \alpha_3 |01\rangle + \alpha_0 |10\rangle + \alpha_1 |11\rangle \tag{22}$$

The single qubit NOT operator already defined in equation (18) can be 'lifted' to an $n$-qubit system, by using a tensor product. So the 2-qubit operator that acts as a NOT on the first qubit, and the identity on the second, is

$$\begin{aligned} N \otimes I_2 &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & 1 \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ 1 \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & 0 \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{aligned} \tag{23}$$

Similarly, the 2-qubit operator that acts as a NOT on the second qubit, and the identity on the first, is $I_2 \otimes N$. (More work is needed to lift $n$-qubit operations to $m > n$ qubit systems if they are not applied to $n$ contiguous qubits in the $m$ qubit space, but the principle of tensor products is still used.)

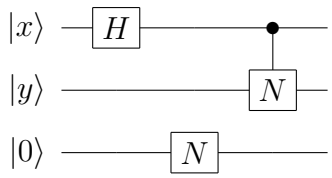



Figure 19: An example quantum circuit diagram

A quantum circuit is a sequence of quantum operations (or quantum 'gates') that act on an initial quantum state to produce the final quantum state. A common way to represent this is using a quantum *circuit diagram*. Each qubit is represented as a horizontal 'wire', and the unitary operations as 'gates' on the relevant wires, read from left to right. If a qubit is not acted on by a gate, there is an implicit tensor product with the identity transform on that qubit. So, consider a three-qubit example, where the initial state of the system is $|0yx\rangle$, and is operated on by a circuit that applies a Hadamard operator to the first qubit, then a NOT to the third, then a controlled not to the second controlled by the first. In matrix form, this is

$$(CN \otimes I_2)(I_4 \otimes N)(H \otimes I_4)\,|0yx\rangle \tag{24}$$

The corresponding quantum circuit diagram form is shown in figure 19. The diagram makes it easier to see what operations are being applied to what qubits. However, care should be taken in reading such diagrams, in particular, in *not* assuming that the wires hold the 'values' of individual qubits. In some cases it is possible to express the final state as a tensor product of individual qubit states, 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. However, in general 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.

# B Quantum Algorithms

There are very few distinct quantum algorithms yet known. Indeed, it was this observation that prompted the authors to engage in using search techniques to look for quantum algorithms in the first place. This appendix gives an overview of known quantum algorithms that are often the target of evolutionary search.

## B.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 was the Deutsch-Josza promise algorithm [79]. Suppose a black box calculates the value of a Boolen function $f(x)$ over a range $x = 0 \ldots 2^n - 1$, 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 $0 \ldots 1$, 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

$$|\Psi\rangle = \tfrac{1}{2}\left( |0\rangle\left(|0\rangle - |1\rangle\right) + |1\rangle\left(|0\rangle - |1\rangle\right)\right) \tag{25}$$

Now apply the unitary function $U_f$ defined by

$$U_f\,|x\rangle\,|y\rangle = |x\rangle\,|y \oplus f(x)\rangle \tag{26}$$

to give

$$\begin{aligned} U_f\,|\Psi\rangle &= \tfrac{1}{2}\left( |0\rangle\left(|f(0)\rangle - |1 \oplus f(0)\rangle\right) + |1\rangle\left(|f(1)\rangle - |1 \oplus f(1)\rangle\right)\right) \\ &= \tfrac{1}{2}\left((-1)^{f(0)}\,|0\rangle\left(|0\rangle - |1\rangle\right) + (-1)^{f(1)}\,|1\rangle\left(|0\rangle - |1\rangle\right)\right) \end{aligned} \tag{27}$$

Now we have

$$U_f\,|\Psi\rangle = \begin{cases} \pm\frac{1}{2}\left(|0\rangle + |1\rangle\right)\left(|0\rangle - |1\rangle\right), & \text{if } f(0) = f(1) \\ \pm\frac{1}{2}\left(|0\rangle - |1\rangle\right)\left(|0\rangle - |1\rangle\right), & \text{if } f(0) \neq f(1) \end{cases} \tag{28}$$

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

$$|\Phi\rangle = \pm\tfrac{1}{\sqrt{2}}\left(|f(0)\rangle \oplus |f(1)\rangle\right)\left(|0\rangle - |1\rangle\right) \tag{29}$$

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.

## B.2 Grover's Algorithm: searching an unstructured database

Consider a function $f$ on the domain $0 \ldots 2^n - 1$ 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 [80], however, demonstrates how a quantum search of $O(\sqrt{2^n})$ is possible.

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

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \tag{30}$$

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

$$U_v |x\rangle = \begin{cases} -|v\rangle, & x = v \\ |x\rangle, & x \neq v \end{cases} \tag{31}$$

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

$$\begin{pmatrix} -1 + \frac{2}{2^N} & \frac{2}{2^N} & \cdots & \frac{2}{2^N} \\ \frac{2}{2^N} & -1 + \frac{2}{2^N} & \cdots & \frac{2}{2^N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{2}{2^N} & \frac{2}{2^N} & \cdots & -1 + \frac{2}{2^N} \end{pmatrix} \tag{32}$$

The application of these two operators in succession is shown in figure 20. Inverting the amplitude of the identified element reduces the average amplitude. When we invert about this new average the amplitude of the identified

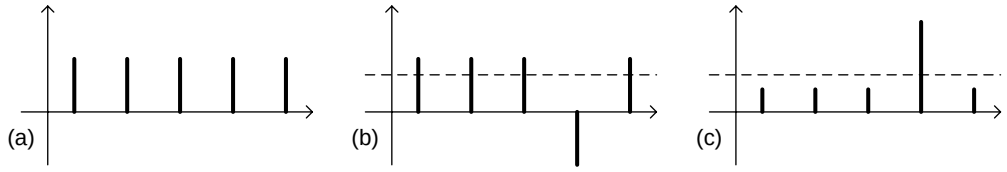



Figure 20: Amplitudes during execution of Grover's algorithm: (a) superposition of states (b) after inversion of identified element (c) after inversion about average value (dashed line)

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 magnitude. 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 [8].) A summary of Grover's algorithm can be found in [81].

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').

## B.3 Shor's Quantum Discrete Fourier Transform

In 1994 Peter Shor's Quantum Discrete Fourier Transform (QDFT) [9] 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 $19^{th}$ century 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

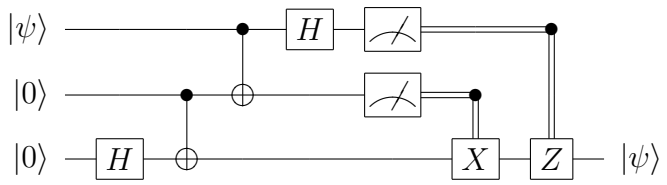



Figure 21: Teleportation circuit

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 [8, Chapter 5].)

## B.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 21.

The first two gates (Hadamard and controlled-NOT) place the second and third qubits in a *maximally entanged* state. 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

$$|\Psi\rangle = a\,|0\rangle + b\,|1\rangle \tag{33}$$

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

$$\begin{aligned} |\Phi\rangle \;=\; & \tfrac{1}{2}\big(\,|00\rangle\,(a\,|0\rangle + b\,|1\rangle) + |01\rangle\,(a\,|1\rangle + b\,|0\rangle) \\ & + |10\rangle\,(a\,|0\rangle - b\,|1\rangle) + |11\rangle\,(a\,|1\rangle - b\,|0\rangle)\big) \end{aligned} \tag{34}$$

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 measurenment results ($M1$ and $M2$) Bob can uses this information to apply a suitable inverse transformations to his third entangled qubit to recover the value of the first source qubit.

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

$$|\Phi\rangle = |01\rangle\,(a\,|1\rangle + b\,|0\rangle) \tag{35}$$

By applying the transformation $X$ to the his (third) qubit Bob can recreate the initial $\Psi$, since

$$X \begin{pmatrix} b \\ a \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} b \\ a \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} \tag{36}$$

If, on the other hand, Bob is informed that two measured values were $|11\rangle$ he can deduce that the remaining state is

$$|\Phi\rangle = |11\rangle \left(a\,|1\rangle - b\,|0\rangle\right) \tag{37}$$

Then by applying the transformation $X$ followed by $Z$, Bob can recreate the initial $\Psi$, since

$$ZX \begin{pmatrix} -b \\ a \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} -b \\ a \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} \tag{38}$$

The general solution is to apply $X^{M2}$ followed by $Y^{M1}$, 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. (It is a feature of quantum information that it cannot be copied. This is the celebrated 'no-cloning' theorem.) 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

$$z := x\,; \quad x := y\,; \quad y := z\,; \tag{39}$$

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

$$x := x \oplus y\,; \quad y := x \oplus y\,; \quad x := x \oplus y\,; \tag{40}$$

Starting with the classical circuit for this program, Mermin [82] carries out justified replacements to derive a quantum analogue, 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.

# C Evolutionary Computation

This appendix gives some background and terminology on search and evolutionary algorithms in general. Michalewicz & Fogel [83] provide an excellent introduction to a range of modern heuristic search techniques.

## C.1 Search terminology

### C.1.1 Solution space and objective function

The *solution space* $\Sigma$ is the space comprising the real world artefacts of interest. In our case it is the space of quantum circuits, quantum protocols, and so on.

The *objective function* $\phi : \Sigma \to \Re$ measures the real world property to be optimised, such as accuracy, efficiency, circuit size, and so on. $\phi$ maps each element of the solution space to a real number that expresses how 'good' it is, in terms of the real world property.

If there are multiple objectives (such as simultaneous high accuracy and small circuit size), $\phi$ 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 $\phi$. From now on, we assume a scalar objective function.

### C.1.2 Search space

Before optimal solutions can 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* $\Gamma : S \to \Sigma$, that maps the search space to the solution space, may be simple, or extremely complicated. For example, interesting work is being done on representation functions that correspond to a 'developmental' phase that 'grows' from a search space 'seed' to the solution space 'organism' [84].

The simplest, and possibly most common, choice of search space is bit strings of length $n$, $S = \{0, 1\}^n$, that directly encode the parameter values being optimised as a binary value. More structured strings of integers and characters may be used, for example, encoding gate type and parameters.

The search space may comprise computer programs, that on execution produce (a description of) a candidate quantum circuit. Execution can be thought of as application of the representation function $\Gamma$.

### C.1.3 The fitness landscape

The *evaluation function* $f : S \to \Re$ evaluates each element of the search space. 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 $\phi$: 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 there may be no other useful relationship between the search and solution spaces. But it 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.

### C.1.4 Evaluation

For simple representations, the fitness of a candidate solution $s$ can be evaluated directly, in terms of the appropriate fitness function, as $f\,s$.

In the case where there is a large distance between the search space and solution space, for example in the case of searching ove the space of 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, as $\phi\,\Gamma\,s$. So every candidate solution has to be 'grown'.

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

### C.1.5 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 (for example, hill climbing, simulated annealing) consider a single search point $s$ at each iteration step, and generate a trace, or *trajectory*, of (search point, evaluation result) pairs $T_t = \langle (s_0, r_0), \ldots (s_t, r_t) \rangle$, where $r_i = f\,s_i$.

Population-based algorithms (for example, evolutionary algorithms, swarm algorithms, immune algorithms) consider a vector of search points $\mathbf{s}$ at each step, and generate a trajectory of sets of (search point vector, evaluation result vector) pairs $T_t = \langle (\mathbf{s}_0, \mathbf{r}_0), \ldots (\mathbf{s}_t, \mathbf{r}_t) \rangle$. (The length of each vector may be a function of the iteration step, if the population size can vary.)

The heart 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, $M : T \to S^k$ (where $k$ is the population size). Commonly, the move function is memoryless; it is a function of only the current state $(\mathbf{s}_t, \mathbf{r}_t)$, 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, $\mathbf{s}_0$. This is often a random start state, or may be seeded with 'good' known solutions, especially in hybrid searching combining several algorithms. 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.

## C.2 Evolutionary algorithms in general

The biological process of evolution provides the metaphor for evolutionary search algorithms (EAs). The search space comprises a population of 'chromosomes', encoding candidate solutions. Each chromosome has several fields,

with values called 'alleles'. 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.

Initialisation and evaluation are covered in the generic search terminology; what moslty distinguishes evolutionary algorithms is their selection and breeding compnents.

### C.2.1 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.

### C.2.2 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 types in the chromosome. 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 tree-based crossover operators, because of the non-linear nature of the artificial chromosome being manipulated. However, it still conforms to the original abstract concept of "*inheritance + variation + selection = evolution*", despite its distance from the biological realisation of this concept.

### C.2.3 Stopping condition

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

The search stops if a good enough solution has 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.

### C.2.4 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 choices 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 was 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 [85].

## C.3 Evolutionary Strategies and Evolutionary Programming: the early days

Some of the earliest work on EAs is known as "Evolutionary Strategies" (ES) [86, 87]. It is characterised by using real-valued chromosomes, directly representing solution space values of interest. Originally ES 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.

Another very early form of EA is known as "Evolutionary Programming" (EP) [88]. It is rather similar to ES, in that it also uses self-adapting mutation parameters, and only mutation. The original application was to evolve finite state automata to recognise and predict strings.

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.

## C.4 Genetic Algorithms: incorporating crossover

Genetic Algorithms (GAs) were invented by John Holland [89, 68], but did not receive that much prominence until they were promoted by his student David Goldberg [90, 91]. Mitchell [92] provides a good introduction to GAs.

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 the GA is well-analysed, and its performance characteristics explained in terms of the schema theorem and

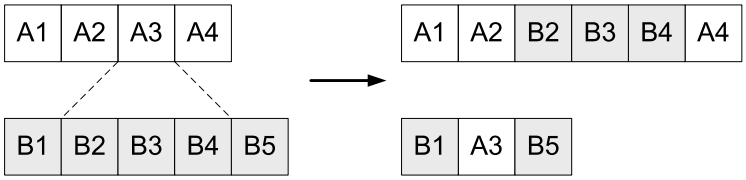



Figure 22: Flexible crossover with list representation

the related $k$-armed bandit theory [89], and the building block hypothesis [90]. (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 [93], even though it cannot be using building blocks of correlated allele values.) More recently, Nix & Vose [94] have analysed GA performance using Markov chains.

## C.5 Genetic Programming

Although there are earlier variants, the first major use of Genetic Programming (GP) was due to John Koza [95]. GP is a variant of GA where the chromosomes are computer programs. It is of particular relevance to evolving quantum programs. Descriptions and applications of GP can be found in the series of books by Koza [95, 96, 97, 98], and also the collection by Kinnear [99]. Banzhaf *et al* [100] provide a good introduction to GP.

The evolved program is usually represented as a tree structure, corresponding to an instance of the parse tree of the programming language. 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 correct, and preferably type-correct, after such surgery. Lisp (for example [101]) 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.

*Linear genetic programming* typically uses varying length lists of imperative language instructions [100]. 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. Figure 22 illustrates a linear GP crossover operation. Linear GP with classical program evolution also allows single operations to be skipped over via preceding branching instructions.

In the simplest variant of GP, each chromosome is a program that is

executed to generate a specific candidate solution. These programs tend to be input-free and have a single behaviour, and so can be evaluated simply by executing them.

In the more general case, the program being evolved is expected to work on a range of inputs. For example, 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.