Human-Competitive Evolution of Quantum Computing Artefacts by Genetic Programming

Paul Massey psm111@cs.york.ac.uk
John A. Clark jac@cs.york.ac.uk
Susan Stepney susan@cs.york.ac.uk
Department of Computer Science, University of York, York, YO10 5DD, UK

Abstract
We show how Genetic Programming (GP) can be used to evolve useful quantum computing artefacts of increasing sophistication and usefulness: firstly specific quantum circuits, then quantum programs, and finally system-independent quantum algorithms. We conclude the paper by presenting a human-competitive Quantum Fourier Transform (QFT) algorithm evolved by GP.

Keywords
quantum computing, genetic programming

1 Introduction

Quantum computing is a radical new computing paradigm that has the potential to bring a new class of previously intractable problems within the reach of computer science (Deutsch, 1985; Rieffel and Polak, 1998; Nielsen and Chuang, 2000). Harnessing the quantum mechanical phenomena of *superposition* and *entanglement*, a quantum computer can perform certain operations exponentially faster than classical (i.e. non-quantum) computers. However, devising algorithms to harness the power of a quantum computer has proved extraordinarily difficult, and it is generally agreed that there are still very few distinct quantum algorithms. This motivates our investigation of Genetic Programming (GP) as a means of discovering new quantum circuits, programs, and ultimately algorithms. GP has discovered new artefacts in other domains. Indeed, its use has produced various patentable outputs. Can it exhibit human-competitive performance for quantum algorithm design?

In this paper we show how GP can evolve quantum circuits to perform specific arithmetic operations. We then show how, by increasing the level of abstraction, it is possible to evolve quantum algorithms capable of solving more generic problems, parameterised by the system size. We present a Quantum Fourier Transform (QFT) algorithm which, when parameterised with a specific system size, generates a circuit that implements the QFT on that size of quantum system. We believe this is the most significant quantum artefact yet evolved using evolutionary computing, and a result which competes with the efforts of human quantum algorithm researchers.

This paper assumes a knowledge of basic GP concepts and techniques. Readers unfamiliar with GP should consult (Koza, 1992; Koza, 1994). A brief overview of the quantum concepts used in this paper is given in Appendix A.

After this introduction, and a brief explanation of our terminology, we review current applications of heuristic search techniques to the design and exploration of

©200X by the Massachusetts Institute of Technology

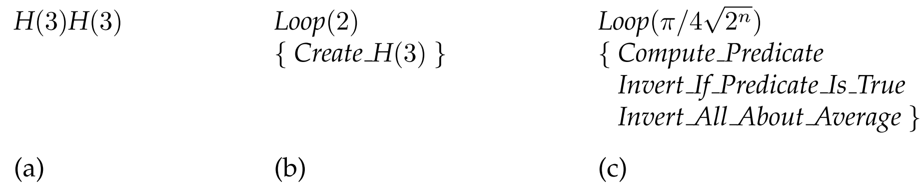

Figure 1: (a) a quantum circuit containing two $H(3)$ gates (i.e. Hadamard gates acting on the third qubit of a particular quantum system: see Appendix B for formal definitions of all quantum gates used in this paper). (b) one possible quantum program (in pseudo-code) which would generate the circuit shown in (a). (c) a true quantum algorithm (actually a key part of Grover's search algorithm (Grover, 1996)), also in pseudo-code. As the parameter n varies, this generates a number of distinct quantum circuits, each of which is able to solve the problem for that particular value of n.

quantum artefacts. We then present the software framework which we have used to evolve quantum artefacts, indicating how solutions are represented and manipulated, as well as presenting the basic approach to evaluating the fitness of individuals. We then present our results at the quantum circuit, quantum program and quantum algorithm levels, before concluding and suggesting some avenues for further work.

2 Terminology

We use the following terminology in this paper:

- *Quantum circuit*: a sequence of quantum instructions (logic gates) that can be applied to a specified quantum system. (See Appendix A for an example.)
- *Quantum program*: a set of (classical) instructions that, when executed, generates (the description of) one or more quantum circuits. The program may include constructs such as iteration and branching functions as well as functions to generate particular quantum gates. In the language of GP, an evolved quantum program is the *genotype* that can be decoded to produce a quantum circuit *phenotype*.
- *Quantum algorithm*: a *parameterisable* quantum program that, as the value of the parameter(s) are altered, generates quantum circuits to solve a large number of different problem instances on different sizes of quantum system.

Figure 1 illustrates the differences between these three concepts. Our terminology differs from normal non-quantum usage where, for example, a (compiled) 'program' is software executed in real circuitry, and where an 'algorithm' is generally a machine-independent recipe that would be implemented by a program.

3 Evolving Quantum Artefacts – A Brief Review

The use of GP to evolve quantum artefacts has been pioneered by Spector *et al.*, who demonstrate evolved circuits to solve instances of OR, AND-OR, Deutsch Josza promise and database search problems (Spector et al., 1998; Spector et al., 1999a; Spector et al., 1999b; Barnum et al., 1999; Barnum et al., 2000). The reader is referred to (Spector, 2004) for a summary and up-to-date discussion of this work.

(Leier and Banzhaf, 2003) use a linear tree GP variant to evolve solutions to the 1-sat problem (Hogg's algorithm). (Williams and Gray, 1999) and (Yabuki and Iba, 2000)

use GP to evolve circuits to implement quantum teleportation. (Spector and Bernstein, 2003) use GP to discover the communications capabilities of quantum circuitry, simultaneously disproving certain conjectures on the communications capacities of quantum channels. It would appear that uncovering genuine insights in this field is computationally tractable using evolutionary computation and the area seems highly promising. Of further interest is that protocols and circuits uncovered by evolutionary computing were generalised by intelligent reflection.

Other researchers have evolved efficient implementations for common quantum gates, on a variety of possible quantum computing architectures. For example, (Gershenfeld and Chuang, 1997) show how the controlled-NOT gate can be implemented on a Nuclear Magnetic Resonance (NMR) based quantum computer by a series of five yet more primitive operations. (Van Meter and Binkley, 2004) outline work in progress seeking to design quantum circuits which optimise the use of quantum resources (for example, operating only on qubits which are physically close together).

A fuller account can be found in (Stepney and Clark, 2006).

4 Q-PACE: Software for Evolving Quantum Artefacts

Our research has been conducted using successive versions of the software suite Q-PACE (**Q**uantum **P**rograms **A**nd **C**ircuits through **E**volution). The original Q-PACE suite is now obsolete. Q-PACE II is a GP suite designed to evolve quantum circuits, Q-PACE III is a development of Q-PACE II designed to evolve quantum programs, and Q-PACE IV is an enhanced version of QPACE III designed to evolve true quantum algorithms. The key properties of these three software suites are described in Appendix D. For convenience, we will use the term “the Q-PACE software” to refer to all three software suites.

An n qubit system forms a 2^n dimensional space, spanned by 2^n orthonormal basis vectors $|k\rangle$ (see Appendix A for more details). 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 α_k are complex *probability amplitudes*.

In order to determine the fitness of an evolved individual circuit U, Q-PACE compares the state vectors generated by applying that individual to a set of known initial states, with those produced by applying a known model solution U_s to the same set of known initial states. More specifically, the technique for assessing fitness is as follows:

1. Initialisation:
   - (a) Create a set $\{|I_i\rangle\}$ of N input state vectors that at least span the space of all possible inputs. Hence $N \geq 2^n$. We take the $\{|I_i\rangle\}$ to be the 2^n basis vectors $|k\rangle$ along with a further 2^n randomly generated vectors. Each member of the set, $|I_i\rangle$, acts as a *fitness case* for the problem under test.
   - (b) Create a set $\{|T_i\rangle\}$ of target vectors, the desired results for each fitness case. Let $|T_i\rangle = \sum_{k=0}^{2^n-1} \tau_k^{(i)} |k\rangle = U_s |I_i\rangle$.
2. Evaluation:
   - (a) Apply the candidate circuit U to each fitness case, to produce a set of *result vectors* $\{|R_i\rangle\}$. Let $|R_i\rangle = \sum_{k=0}^{2^n-1} \rho_k^{(i)} |k\rangle = U |I_i\rangle$.

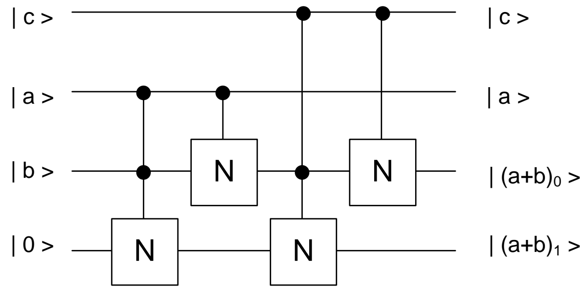

Figure 2: evolved deterministic adder circuit

(b) Compare the result vector set $\{|R_i\rangle\}$ with the corresponding target vector set $\{|T_i\rangle\}$. The chosen means of comparison defines the specific *fitness function* for the particular problem under test.

5 Evolving Quantum Circuits: Adders

Q-PACE II has used a number of different fitness functions suitable for the evolution of various different kinds of quantum circuits and algorithms.

5.1 Deterministic Quantum Adders

A typical fitness function might sum the magnitudes of the differences of the probability amplitudes of each state vector:

$$f_g = \sum_{i=1}^{N} \sum_{k=0}^{2^n-1} g\left(|\tau_k^{(i)} - \rho_k^{(i)}|\right) \tag{1}$$

Q-PACE II is able to evolve a range of deterministic arithmetic circuits, including 2-bit half- and full-adders. For example, using the fitness function f_g with $g(x) = x$, a population size of 200, a crossover probability of 0.5, and a mutation probability of 0.01, Q-PACE II evolved (in 943 generations) an exact quantum full adder circuit (Massey et al., 2004), see figure 2. As far as we are aware, this is the most efficient quantum full-adder that can be generated with the quantum gates available to Q-PACE II. However, it is not a new discovery, since it was first published in (Gossett, 1998).

5.2 Probabilistic Quantum Adders

A *probabilistic* quantum circuit is defined as one which need not always give the correct answer, as long as the probability of it doing so is at least 50% for every fitness case tested. To evolve probabilistic quantum circuits, we use a fitness function based on earlier work by (Spector et al., 1998; Spector et al., 1999a; Spector et al., 1999b):

$$f_S = \textit{hits} + \textit{correctness} + \textit{efficiency} \tag{2}$$

The *hits* component focusses on getting every fitness case 'good enough'. It is the total number of fitness cases, N, minus the number of fitness cases where the circuit produces the correct answer with a probability of more than $0.5 + \epsilon$ (a non-zero ϵ is chosen to counteract rounding errors, usually $\epsilon = 0.02$). For each correct (target) amplitude vector $|T_i\rangle$ the probability of observing basis state $|k\rangle$ is $|\tau_k^{(i)}|^2$. The probability

of obtaining $|k\rangle$ in the corresponding result vector $|R_i\rangle$ is $|\rho_k^{(i)}|^2$. We say a circuit gives the correct answer a proportion p of the time if, for all k, $|\rho_k^{(i)}|^2 \geq p|\tau_k^{(i)}|^2$.

We define the boolean delta function to be

$$\delta(b) = \text{if } b \text{ then } 1 \text{ else } 0 \tag{3}$$

Then

$$\textit{hits} = N - \sum_{i=1}^{N} \delta\left(\forall k \in \{0..2^n - 1\} : |\rho_k^{(i)}|^2 \geq p|\tau_k^{(i)}|^2\right) \tag{4}$$

The *correctness* component focusses on getting close to the correct answer. Here we are interested in cases where several states may have nonzero probability amplitudes, so there are several potentially correct answers. So the error for each fitness case is defined as the difference between the correct (target) probability of observing basis state $|k\rangle$ minus the result vector probability of observing the basis state $|k\rangle$, summed over all k for which the target vector has a non-zero amplitude.

$$\textit{error}_i = \sum_{k=0}^{2^n-1} \delta\left(\tau_k^{(i)} \neq 0\right) \left||\tau_k^{(i)}|^2 - |\rho_k^{(i)}|^2\right| \tag{5}$$

It is desirable for the fitness function to focus on attaining probabilistically correct answers to *all* fitness cases, rather than reducing the error in those, possibly few, fitness cases where it is already good enough (e.g. reducing the error from 0.45 to 0.40). So only the part of the error greater than $0.5 - \epsilon$ is used in the fitness measure. It is also 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 $\textit{hits} < 1$) before being used in the fitness function. So the *correctness* component is:

$$\textit{correctness} = \frac{\sum_{i=1}^{N} \max\left(0, \textit{error}_i - (0.5 - \epsilon)\right)}{\max(\textit{hits}, 1)} \tag{6}$$

Each solution comprises a number of individual allele gates. We want small circuits, so we impose an efficiency component on the fitness. For a system size of n qubits, let the number of allele gates be S_n, and the target number of gates (for example, the known minimum circuit size) be $\mathcal{T}_n$. We sum the size difference over all the system sizes tested. The *efficiency* component is:

$$\textit{efficiency} = \mathcal{W}_e \sum_n S_n - \mathcal{T}_n \tag{7}$$

where $\mathcal{W}_e$ is the efficiency weighting: it is usually taken to be 0.01.

Using this fitness function f_S, Q-PACE II can find probabilistic solutions to problems for which it was unable to find a deterministic solution. As a typical example, Q-PACE II evolved a probabilistic half-adder on 3 qubits using only the H gate and the zeroing gate Z, together with their controlled equivalents (Massey et al., 2004), see figure 3.

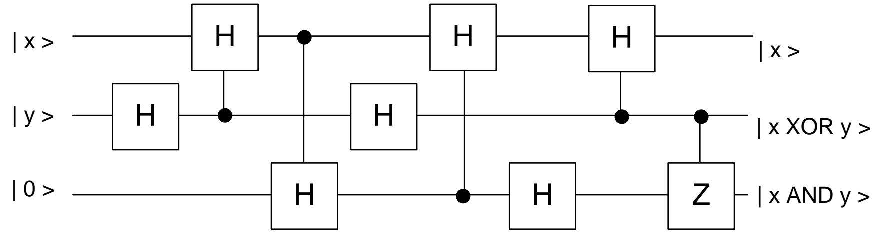

Figure 3: evolved probabilistic half-adder circuit, which delivers the correct result with a probability > 0.5

6 Evolving Quantum Algorithms: The Quantum Fourier Transform

6.1 Definition of the QFT

Consider a quantum state vector $|X\rangle = \sum_{k=0}^{2^n-1} x_k |k\rangle$. Applying the QFT to this state vector gives us a result state vector $|Y\rangle = \sum_{k=0}^{2^n-1} y_k |k\rangle$ where

$$y_k = \frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} x_j \exp \frac{2\pi i \, jk}{2^n} \tag{8}$$

6.2 Implementing the QFT

(Shor, 1994; Shor, 1997) describe an algorithm for implementing the QFT. The following pseudo-code algorithm, $QFT(n)$, captures Shor's algorithm, implementing the QFT on any size of quantum system using only alleles and quantum gates available to Q-PACE IV:

```
For (j = 1; j < system_size; j++) {
    Create_H(j);
    For (k = 1; k <= (system_size - j); k++) {
        Create_CP(j+k, j, k+1); } }
Create_H(system_size);
For (m = 1; m <= (system_size / 2); m++) {
    Create_SWAP(m, system_size - m + 1); }
```

The functions `Create_H(x)`, `Create_CP(x,y,z)` and `Create_SWAP(x,y)` are as defined in Appendix C. Each circuit produced by this algorithm implements an exact QFT for that system size. The circuits produced for quantum systems of between 1 and 5 qubits are

1. $H(1)$
2. $H(1), CP(2,1,\pi/4)$,
   $H(2)$,
   $SWAP(1,2)$
3. $H(1), CP(2,1,\pi/4), CP(3,1,\pi/8)$,
   $H(2), CP(3,2,\pi/4)$,
   $H(3)$,
   $SWAP(1,3)$

4. $H(1), CP(2,1,\pi/4), CP(3,1,\pi/8), CP(4,1,\pi/16),$
   $H(2), CP(3,2,\pi/4), CP(4,2,\pi/8),$
   $H(3), CP(4,3,\pi/4),$
   $H(4),$
   $SWAP(1,4), SWAP(2,3)$

5. $H(1), CP(2,1,\pi/4), CP(3,1,\pi/8), CP(4,1,\pi/16), CP(5,1,\pi/32),$
   $H(2), CP(3,2,\pi/4), CP(4,2,\pi/8), CP(5,2,\pi/16),$
   $H(3), CP(4,3,\pi/4), CP(5,3,\pi/8),$
   $H(4), CP(5,4,\pi/4),$
   $H(5),$
   $SWAP(1,5), SWAP(2,4)$

7 An Evolved Program to Implement the QFT on a 3-qubit system

7.1 Fitness Function

The fitness function used to evolve QFT algorithms of a specific system size is:

$$f_3 = f_g + \mathit{efficiency} \tag{9}$$

The fitness term f_g is defined in equation (1); here we use $g(x) = \delta(x \neq 0)$, which gives credit only for exact matches in the various state vector components. The *efficiency* term is that defined in equation (7). The target size parameter $\mathcal{T}_n$ is here defined as $\mathcal{T}_1 = 2$, $\mathcal{T}_2 = 6$, $\mathcal{T}_3 = 10$, and $\mathcal{T}_4 = 16$. These values are a little greater than the most efficient known QFT circuits for these system sizes.

7.2 The Evolved Program

Q-PACE III is able to evolve programs which, when decoded and executed, implement an exact QFT on a 3 qubit quantum system (Massey et al., 2005). Using the fitness function f_3 of equation (9), a population size of 100, a crossover probability of 0.5, and a mutation probability of 0.01, Q-PACE III evolved (in 1122 generations) the following individual:

```
ROOT(
    2,
    Create_CP( 3, Create_CH(1,1), Create_CP(1,2,2) ),
    2,
    Create_H(2),
    Create_CP(2,3,2),
    Create_CCN(
        Create_CP(
            Create_CCN( Create_CP(3,1,3), Create_N(1), 1 ),
            3, 2 ),
        Create_H(3),
        1 ),
    Create_CN(1,3),
    2, 3 )
```

This individual, when decoded/executed, produces the quantum circuit illustrated in figure 4.

After hand-optimisation to remove the two consecutive NOT gates, the circuit can be simplified to that illustrated in figure 5. This latter circuit has 10 gates. Although the best known circuit to generate QFT(3) can be implemented in 7 gates, that circuit requires the use of a SWAP gate, which was not available as an allele to Q-PACE III in

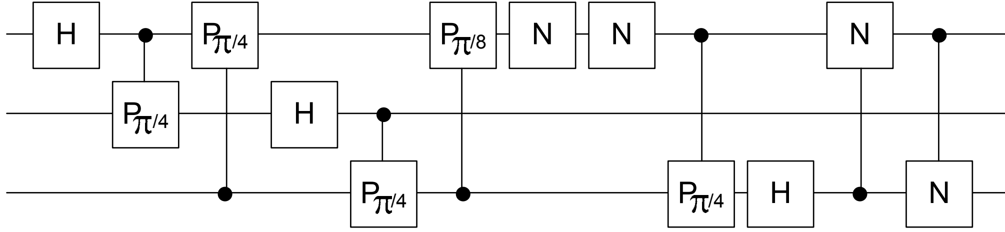

Figure 4: evolved QFT on a 3 qubit quantum system.

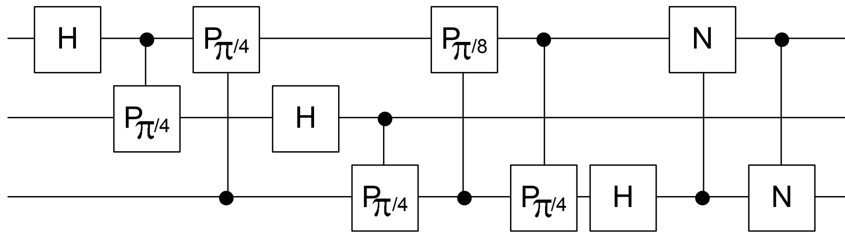

Figure 5: evolved QFT after hand-optimisation.

this particular GP run. The most efficient known circuit to implement QFT(3) using the alleles given to Q-PACE III in this GP run has 9 gates, just one less than the solution evolved here.

8 An Evolved Algorithm to Implement the QFT on system sizes of 1 to 3 qubits

8.1 Fitness Function

We now introduce a new fitness function used to evolve QFT algorithms for general system sizes. Here, we define the target and result vector component amplitudes in polar coordinate form: $\tau_k^{(i)} = (r_{\tau_k}^{(i)}, \theta_{\tau_k}^{(i)})$ and $\rho_k^{(i)} = (r_{\rho_k}^{(i)}, \theta_{\rho_k}^{(i)})$.

The fitness function sums the polar coordinate differences between corresponding state vector components.

$$f_{1\ldots 3} = \alpha \sum_{i=1}^{N} \sum_{k=0}^{2^n-1} |r_{\tau_k}^{(i)} - r_{\rho_k}^{(i)}| + \sum_{i=1}^{N} \sum_{k=0}^{2^n-1} |\theta_{\tau_k}^{(i)} - \theta_{\rho_k}^{(i)}| + \textit{efficiency} \quad (10)$$

This fitness function makes use of a scaling factor α (which we typically set to an integer between 2 and 5), the purpose of which is to ensure that individuals where the magnitude of the complex numbers match (but the phases do not) have a considerably better fitness than individuals where the phases match but the magnitudes do not. The fitness function is designed this way to promote a particular evolutionary strategy: to allow the GP software to first evolve solutions which are basically correct but with incorrect angles in any phase gates ($e.g.$ $CP(2,1,\pi/8)$ instead of $CP(2,1,\pi/4)$), before subsequently evolving the correct angles. We have found this strategy, by and large, works well for solving problems where quantum phase operations are an integral part

```
ROOT(
  ITERATE(
    MINUS(n,1,n),
    BODY(
      Create_H(v1,n,n),
      ITERATE(
        MINUS(n,v1,n),
        BODY(
          Create_CP( PLUS(v1,v2,n), v1, PLUS(1,v2,v2) ),
          v1, 2 ),
        v1 ),
      n ),
    n )
  Create_H(n,n,n),
  ITERATE(
    DIVIDE(n,n,n),
    BODY( Create_SWAP(v1,n,2), DIVIDE(n,n,n), n ),
    n )
)
```

Figure 6: The evolved QFT algorithm, for quantum system size $n = 1 - 3$

of the solution.

The *efficiency* term and corresponding target size parameter $\mathcal{T}_n$ are as for the previous case (equation 9).

8.2 The Evolved Algorithm

Q-PACE IV is able to evolve algorithms which, when decoded and executed, implement an exact QFT on system sizes of 1, 2 and 3 qubits (Massey et al., 2005). One is presented here. To evolve this algorithm, the GP used the fitness function $f_{1\ldots3}$ of equation (10), a population size of 2000 for the first two generations and 50 thereafter (to ensure a "deep gene pool" at the beginning of the evolutionary process), a crossover probability of 0.75, and a mutation probability of 0.075. With these parameters, and testing candidate solutions against system sizes of 1, 2 and 3 qubits, Q-PACE IV evolved (in 2177 generations) the individual shown in figure 6.

This individual, when decoded/executed, produces the following quantum circuits for system size n:

1. $H(1), SWAP(1,1)$

2. $H(1), CP(2,1,\pi/4)$,
   $H(2)$,
   $SWAP(1,2)$

3. $H(1), CP(2,1,\pi/4), CP(3,1,\pi/8)$,
   $H(2), CP(3,2,\pi/4)$,
   $H(3)$,
   $SWAP(1,3)$

4. $H(1), CP(2,1,\pi/4), CP(3,1,\pi/8), CP(4,1,\pi/16)$,
   $H(2), CP(3,2,\pi/4), CP(4,2,\pi/8)$,
   $H(3), CP(4,3,\pi/4)$,
   $H(4)$,
   $SWAP(1,4)$

These circuits are human-competitive for $n = 1 - 3$: there is one redundant gate in the circuit for a 1 qubit system, but the other two circuits equal the most efficient known using these quantum gates.

However, this algorithm does not implement the QFT perfectly for systems with more than three qubits. The last ITERATE loop always runs for precisely one iteration, and therefore there is always precisely one SWAP gate generated, regardless of the system size. For system sizes above 3, multiple SWAP gates are required to implement the QFT exactly (more precisely, $\lfloor n/2 \rfloor$ gates are needed, where n is the system size). The $n = 4$ circuit shown is a reliable $QFT(4)$ circuit apart from a missing $SWAP(2,3)$ gate at the end. This algorithm becomes increasingly poor at implementing the QFT as the system size increases.

9 An Evolved Algorithm to Implement the QFT on any size of quantum system

With fitness function of equation (10), and allowed to test candidate solutions against system sizes of 1, 2, 3 and 4 qubits, Q-PACE IV is unable to evolve an algorithm that implements the QFT operation exactly on an arbitrary size of quantum system.

We introduce an additional term $swap$ to penalise the absence of an appropriate number of SWAP gates for the system size under consideration.

$$f_n = f_{1\ldots3} + \mathit{swap} \tag{11}$$

$$\mathit{swap} = \mathcal{W}_n \sum_n \delta\left(\#\text{SWAP}_n \neq \left\lfloor \frac{n}{2} \right\rfloor\right) \tag{12}$$

In our work the weighting factor $\mathcal{W}_n = 50 \left\lfloor \frac{n}{2} \right\rfloor$.

In this respect we have given the technique a piece of system specific help. It would alternatively be legitimate to evolve a related QFT circuit without these final SWAP gates; this is how the circuit is presented in text books such as (Nielsen and Chuang, 2000), for example.

When set up with this new fitness function, and the same parameters as in section 8, but allowed to test candidate solutions against system sizes of 1, 2, 3 and 4 qubits, Q-PACE IV evolved (in 2436 generations) an algorithm that implements the QFT operation exactly on any size of quantum system (Massey et al., 2005), see figure 7.

This individual, when decoded/executed, produces the following quantum circuits for system sizes $n = 1 - 5$:

1. $H(1)$
2. $H(1), CP(2,1,\pi/4)$,
$H(2)$,
$SWAP(1,2)$
3. $H(1), CP(2,1,\pi/4), CP(3,1,\pi/8)$,
$H(2), CP(3,2,\pi/4)$,
$H(3)$,
$SWAP(1,3)$

```
ROOT(
  ITERATE(
    MINUS(n,1,4),
    BODY(
      Create_H(v1,n,n),
      ITERATE(
        MINUS(n,v1,v1),
        BODY(
          Create_CP( PLUS(v1,v2,v1), v1, PLUS(v2,1,4) ),
          1, 1 ),
        v1 ),
      1 ),
    n ),
 Create_H(n,1,n),
 ITERATE(
   DIVIDE(n,2,n),
   BODY(
     Create_SWAP( v1, PLUS( MINUS(n,v1,1), 1, 3 ), 1 ),
     n, n ),
   3 )
)
```

Figure 7: The evolved QFT algorithm, for any quantum system size

4. $H(1), CP(2,1,\pi/4), CP(3,1,\pi/8), CP(4,1,\pi/16),$
   $H(2), CP(3,2,\pi/4), CP(4,2,\pi/8),$
   $H(3), CP(4,3,\pi/4),$
   $H(4),$
   $SWAP(1,4), SWAP(2,3)$

5. $H(1), CP(2,1,\pi/4), CP(3,1,\pi/8), CP(4,1,\pi/16), CP(5,1,\pi/32),$
   $H(2), CP(3,2,\pi/4), CP(4,2,\pi/8), CP(5,2,\pi/16),$
   $H(3), CP(4,3,\pi/4), CP(5,3,\pi/8),$
   $H(4), CP(5,4,\pi/4),$
   $H(5),$
   $SWAP(1,5), SWAP(2,4)$

As can be seen from carefully comparing the algorithmic form of figure 7 with the pseudocode in section 6.2, it correctly generates a QFT circuit for any system size n. These circuits are human-competitive: each one equals the most efficient known circuit for that system size.

10 Discussion

Q-PACE II has evolved deterministic and probabilistic 2-bit adder circuits.

Q-PACE III has evolved a program that generates a 3 qubit QFT circuit (figure 5). Although nearly as efficient as the best known circuit, it is rather irregular in structure. It shows no clear pattern that could be used as the basis of a general circuit.

Q-PACE IV has evolved an algorithm that generates a QFT circuit for system sizes 1, 2, and 3 (section 8), by testing against systems sizes of 1, 2 and 3 qubits. It does not generalise to larger system sizes, however, having too few SWAP gates in those cases. There is insufficient information in the supplied test cases to allow such generalisation. The evolved program generates a single swap gate, SWAP$(1,n)$, which is the minimum

adequate for the supplied test cases, but is not adequate for larger systems. The 'core' part of the evolved QFT (the part before the swap gates) does generalise, however.

Q-PACE IV has also evolved an algorithm that generates a QFT circuit for arbitrary system sizes (section 9), by testing against system sizes of 1, 2, 3 and 4 qubits. Despite being evolved against only small test systems, it successfully generalises to larger systems sizes. There is enough information present in this larger set of test cases for the evolutionary process to determine the general solution.

The circuits so generated exactly reproduce the previously known QFT circuits, and so, in particular, exhibit regular structures. It seems that the most efficient way to discover solutions for a range of system sizes is to evolve a general algorithm capturing the underlying structure. This suggests that it might be more fruitful to try to evolve algorithms for a range of system sizes, rather than the apparently "easier" problem of evolving for a specific system size.

11 Conclusions

In this paper, we show that GP can evolve system size-independent quantum algorithms capable of generating a correct (and efficient) circuit for any supplied system size.

However, the results presented here do not extend the portfolio of known quantum algorithms. Given the difficulty of devising new quantum algorithms analytically, an important open research problem remains: can GP evolve new quantum algorithms to solve open problems in computer science?

12 Acknowledgements

Our thanks to Riccardo Poli for detailed comments on an earlier version of this paper, and especially for the suggestion to use a boolean delta function in order to simplify the presentation of many of the fitness functions.

References

Barnum, H., Bernstein, H. J., and Spector, L. (1999). A quantum circuit for OR. *LANL pre-print quant-ph/990756*.

Barnum, H., Bernstein, H. J., and Spector, L. (2000). Quantum circuits for OR and AND of ORs. *J. Physics A: Mathematical and General*, 33(45):8047–8057.

Deutsch, D. (1985). Quantum theory, the Church-Turing thesis, and the Universal Quantum Computer. *Proc. Roy. Soc. London, series A*, 400:97.

Gershenfeld, N. A. and Chuang, I. L. (1997). Bulk spin-resonance quantum computing. *Science*, 275:350–356.

Gossett, P. (1998). Quantum carry-save arithmetic. *LANL pre-print quant-ph/9808061*.

Grover, L. (1996). A fast quantum mechanical algorithm for database search. In *Proc. 28th ACM STOC*, page 212.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press.

Leier, A. and Banzhaf, W. (2003). Evolving Hogg's quantum algorithm using linear-tree GP. In *Proc. GECCO 2003*, volume 2723 of *LNCS*, pages 390–400. Springer.

Massey, P., Clark, J. A., and Stepney, S. (2004). Evolving quantum programs and circuits through genetic programming. In *Proc. GECCO 2004*, volume 3103 of *LNCS*, pages 569–580. Springer.

Massey, P., Clark, J. A., and Stepney, S. (2005). Evolution of a human-competitive quantum fourier transform algorithm using genetic programming. In *Proc. GECCO 2005*, pages 1657–1664. ACM Press.

Nielsen, M. A. and Chuang, I. L. (2000). *Quantum Computation and Quantum Information*. Cambridge University Press.

Rieffel, E. and Polak, W. (1998). An introduction to quantum computing for non-physicists. *LANL pre-print quant-ph/9809016*.

Shor, P. W. (1994). Algorithms for quantum computation : Discrete logarithms and factoring. In *Proc. 35th IEEE Symp. on the Foundations of Computer Science*, page 124.

Shor, P. W. (1997). Polynomial time algorithms for prime-factorisation and discrete logarithms on a quantum computer. *SIAM Journal of Computing*, 26:1484.

Spector, L. (2004). *Automatic Quantum Computer Programming: a genetic programming approach*. Kluwer.

Spector, L., Barnum, H., and Bernstein, H. (1998). Genetic Programming for quantum computers. In *Genetic Programming 1998 – Proceedings of the Third Annual Conference*. Morgan Kaufmann.

Spector, L., Barnum, H., Bernstein, H., and Swamy, N. (1999a). Finding a better-than-classical quantum AND/OR algorithm using Genetic Programming. In *Proc. 1999 Congress on Evolutionary Computation*.

Spector, L., Barnum, H., Bernstein, H., and Swamy, N. (1999b). Quantum computing applications of Genetic Programming. In *Advances in Genetic Programming 3*. MIT Press.

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

Stepney, S. and Clark, J. A. (2006). Evolving quantum programs and protocols. In Rieth, M. and Schommers, W., editors, *Handbook of Theoretical and Computational Nanotechnology*, chapter 90. American Scientific Publishers.

Van Meter, R. and Binkley, K. (2004). Compiling quantum programs using genetic algorithms, discovered in part through Genetic Programming. In *The Wild and Crazy Idea Session IV, abstracts, part of 11th Intl. Conf. Architectural Support for Programming Languages and Operating Systems, October 2004*.

Wall, M. (2005). GALib, a C++ library for Genetic Algorithms. *Available from* `http://lancet.mit.edu/ga/`.

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

Yabuki, T. and Iba, H. (2000). Genetic algorithms for quantum circuit design – evolving a simpler teleportation circuit. In *Late Breaking Papers at GECCO 2000*, pages 425–430.

References of the form "LANL pre-print *quant-ph/xxxxxxx*" are available on the Internet from the Los Alamos National Laboratory pre-print server at `http://www.arXiv.org`

A Brief review of 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 (Rieffel and Polak, 1998). 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 α and β 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 $|\beta|^2$. α and β are complex *probability amplitudes*.

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

$$|0\rangle \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \quad |1\rangle \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{13}$$

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \equiv \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{14}$$

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

$$N \equiv \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}; \quad N\,|\Psi\rangle \equiv \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix} \equiv \beta\,|0\rangle + \alpha\,|1\rangle \tag{15}$$

Some further quantum operations are given in Table 1. 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 α_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

$$|\Phi\rangle = \alpha_0 |00\rangle + \alpha_1 |01\rangle + \alpha_2 |10\rangle + \alpha_3 |11\rangle \equiv \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} \tag{16}$$

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.

$$CN = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{17}$$

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:

$$\alpha_0 |00\rangle + \alpha_1 |01\rangle + \alpha_2 |10\rangle + \alpha_3 |11\rangle \xrightarrow{NOT} \alpha_2 |00\rangle + \alpha_3 |01\rangle + \alpha_0 |10\rangle + \alpha_1 |11\rangle \tag{18}$$

The single qubit NOT operator already defined in equation (15) 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

$$\begin{aligned} N \otimes I_2 &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & 1 \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ 1 \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & 0 \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{aligned} \tag{19}$$

Similarly, the 2-qubit operator that acts as a NOT on the second qubit, and the identity on the first, is $I_2 \otimes N$. (A little fancy footwork 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.)

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

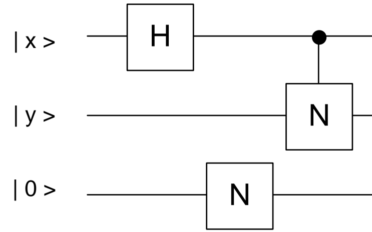

Figure 8: An example quantum circuit diagram

name	*symbol*	U
NOT	$N(x)$	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Hadamard	$H(x)$	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
Rotation about x	$RX(x, 2\theta)$	$\begin{pmatrix} \cos\theta & -i\sin\theta \\ -i\sin\theta & \cos\theta \end{pmatrix}$
Rotation about y	$RY(x, 2\theta)$	$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$
Rotation about z	$RZ(x, 2\theta)$	$\begin{pmatrix} \exp -i\theta & 0 \\ 0 & \exp i\theta \end{pmatrix}$
V gate	$V(x)$	$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$
W gate	$W(x)$	$\begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix}$
Zeroing gate	$Z(x)$	(non-unitary)

Table 1: The quantum gates available to Q-PACE II

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

$$(CN \otimes I_2)(I_4 \otimes N)(H \otimes I_4)\,|0yx\rangle \tag{20}$$

The corresponding quantum circuit diagram form is shown in figure 8. 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 Gates available to Q-PACE

Table 1 shows the quantum gates available to Q-PACE II. The following points should be noted:

name	*symbol*	U
Phase	$P(x, 2\theta)$	$\begin{pmatrix} 1 & 0 \\ 0 & \exp i\theta \end{pmatrix}$
Pauli Y transform	$Y(x)$	$\begin{pmatrix} 0 & i \\ i & 0 \end{pmatrix}$
SWAP	SWAP(x, y)	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Table 2: The additional quantum gates available to Q-PACE III and IV

1. *Controlled* versions of each of the above gates are also available. As a typical example, the Controlled Hadamard gate $H(c, x)$ has the same effect as the Hadamard gate $H(x)$ if the value of qubit $c \neq 0$, otherwise it has no effect.
2. The *RX*, *RY* and *RZ* gates may take four possible values for θ, namely 1, π, $\pi/2$ and $\pi/4$ radians.
3. The zeroing gate $Z(x)$ is non-unitary. It is provided as an allele to allow qubits to be initialised to zero, but evolved circuits can make use of Z in mid-circuit (Massey et al., 2004). One way to implement Z in mid circuit is to $SWAP$ it with an ancilla qubit initialised to zero.

Table 2 shows the additional quantum gates available to Q-PACE III and IV. The following point should be noted:

1. As well as controlled versions of all basic gate types, a *double-controlled* NOT gate, $CCN(c1, c2, x)$, is also available to Q-PACE III and IV (equivalent to the gate $N(x)$ if neither qubit $c1$ nor qubit $c2$ have a value of zero, otherwise has no effect).

C Q-PACE IV Allele Set

Table 3 shows the gate generating functions used in Q-PACE IV. The following points should be noted:

1. All the functions in this set return the value x.
2. If any parameter q that represents a qubit position has a value that falls outside the system size n, it is coerced: if $q < 1$, then q is coerced to 1; if $q > n$, then q is coerced to n.

Table 4 shows the other alleles used in Q-PACE IV.

D Comparison of the Q-PACE II, III and IV software suites

- *Language and GP Engine*
  - All: C++ software suites, with GP engines based on (Wall, 2005)'s GALib library
- *Representation*

name	*side effect*
Create_$N(x)$	create $N(x)$
Create_$CN(c,x)$	if $c \neq x$, create $CN(c,x)$; else create $N(x)$
Create_$CCN(c1,c2,x)$	if $c1 \neq c2 \neq x$, create $CCN(c1,c2,x)$ if $c1 = c2 \neq x$, create $CN(c1,x)$ if $c1 = x \neq c2$, create $CN(c2,x)$ if $c1 \neq x = c2$, create $CN(c1,x)$ if $c1 = c2 = x$, create $N(x)$
Create_$H(x)$	create $H(x)$
Create_$CH(c,x)$	if $c \neq x$, create $CH(c,x)$; else create $H(x)$
Create_$P(x,\theta)$	create $P(x,\pi/2\theta)$
Create_$CP(c,x,\theta)$	if $c \neq x$, create $CP(c,x,\pi/2\theta)$; else create $P(x,\pi/2\theta)$
Create_$Y(x)$	creates $Y(x)$
Create_$CY(c,x)$	if $c \neq x$, create $CY(c,x)$; else create $Y(x)$
Create_$RX(x,\theta)$	create $RX(x,\pi/2\theta)$
Create_$CRX(c,x,\theta)$	if $c \neq x$, create $CRX(c,x,\pi/2\theta)$; else create $RX(x,\pi/2\theta)$
Create_$RY(x,\theta)$	create $RY(x,\pi/2\theta)$
Create_$CRY(c,x,\theta)$	if $c \neq x$, create $CRY(c,x,\pi/2\theta)$; else create $RY(x,\pi/2\theta)$
Create_$RZ(x,\theta)$	create $RZ(x,\pi/2\theta)$
Create_$CRZ(c,x,\theta)$	if $c \neq x$, create $CRZ(c,x,\pi/2\theta)$; else create $RX(x,\pi/2\theta)$
Create_SWAP(x,y)	if $x \neq y$, create SWAP(x,y); else no effect

Table 3: The gate generating functions used in Q-PACE IV

- Q-PACE II: 1st order: individuals are quantum circuits. Each individual is a list of alleles.
- Q-PACE III: 2nd order: individuals are programs which, when decoded, generate a single quantum circuit appropriate to a single size of quantum system. Each individual is a tree of alleles; a linked list holds the quantum circuit produced when the individual is decoded.
- Q-PACE IV: 2nd order: individuals are pseudo-code algorithms which, when decoded/executed, produce a family of quantum circuits: one for each size of quantum system under test. Each individual is a tree of alleles; a linked list holds the quantum circuit produced when the individual is decoded for a given system size.

• *Allele set*

- Q-PACE II: The set of quantum gates available to the GP suite.
- Q-PACE III: The allele set comprises both *functions* and *terminals*. Each function generates a quantum gate. Each terminal is a constant (denoting which qubit should be operated on).
- Q-PACE IV: The allele set comprises both *functions* and *terminals*. Functions may be *gate-generating functions*, *arithmetic functions* or *control functions*; terminals may be *constants* or *variables*. Formal definitions of all the gate-generating functions and of all other alleles used in Q-PACE IV are presented in Appendix C.

name	*return value*	*comments*
PLUS(x, y)	$x + y$	
MINUS(x, y)	$x - y$	
MULTIPLY(x, y)	$x * y$	
DIVIDE(x, y)	int(x/y)	more precisely, returns int(x/y) if $y \neq 0$, otherwise 1
ITERATE(n, BODY)	n	the second child of an ITERATE function is always a BODY function (enforced during crossover and mutation)
BODY$(ch1, ch2, ch3)$	$ch1$	
ROOT$(ch1, ch2, \ldots)$	$ch1$	all individuals are rooted in this function; it can appear nowhere else in an individual
Plus **Constants** $(1..n)$ and **Variables** (n, plus loop counters v_i for any ITERATE statements currently in scope).		

Table 4: The other alleles used in Q-PACE IV

- *Quantum gate set*
  - Q-PACE II: See Table 1
  - Q-PACE III and IV: See Tables 1 and 2
- *User restriction of allele and quantum gate sets*
  - Q-PACE II: A user of the software may restrict the GP to use a subset of the gates in the allele set by selecting (through an on-screen prompt at the start of a GP run) which specific gates should be used.
  - Q-PACE III and IV: A user of the software may restrict the GP to use any subset of the gates in the library by selecting (through an on-screen prompt at the start of a GP run) which specific gates should be used. The allele set can be similarly restricted.
- *Parameterisation of function alleles*
  - Q-PACE II: N/A: all alleles are quantum gates.
  - Q-PACE III: Functions take variable numbers of parameters.
  - Q-PACE IV: All functions take the same number of parameters (3) to allow more general mutation operators. When a function requires fewer than 3 parameters, the remaining parameters are ignored at execution time.
- *Selection and crossover operators*
  - All: Tournament selection and subtree-swap crossover
- *Mutation operators*
  - Q-PACE II: A range of operators available including gate replacement (a quantum gate allele is replaced by another), gate insertion, gate deletion, and parameter mutation (the target or control qubit can be changed without the gate being changed).

- Q-PACE III: Terminals may mutate into other terminals, functions may mutate into other functions of the same cardinality.
- Q-PACE IV: One of three operators ("mini", "midi" or "maxi") is chosen by biased coin flip. In mini-replace, an allele is mutated for another allele of the same type (e.g. a constant can only mutate into another constant); any children of the original allele are unchanged. In midi-replace, a terminal can mutate into any other terminal (e.g. a variable can become a constant, and vice versa), and a function can mutate into any other function (e.g. a gate producing function can become an arithmetic function, and vice versa); children of the original allele are unchanged. In maxi-replace, an allele can mutate into any other allele. If the original allele has children, they are destroyed and rebuilt at random.

- *Method of allele selection for initial population*
  - Q-PACE II: All individuals in the initial population are of the same length; alleles are chosen at random.
  - Q-PACE III and IV: While *current_tree_depth* $<$ *max_tree_depth*, an allele will be a randomly-selected function with some probability *function_probability*, and a randomly-selected terminal with probability $(1 - \textit{function_probability})$. When *current_tree_depth* reaches *max_tree_depth*, all new alleles are randomly selected terminals.
- *Stopping criteria*
  - All: Evolution continues until either (a) an exact solution to the problem under test is found (in which case it is displayed), or (b) a user-defined number of generations elapse (in which case the best result so far is displayed).