Evolving Quantum Circuits using Genetic Programming

B. I. P. Rubinstein
The University of Melbourne,
Parkeville,
Melbourne, Australia 3052
b.rubinstein@ugrad.unimelb.edu.au

Abstract - This paper presents a new representation and corresponding set of genetic operators for a scheme to evolve quantum circuits with various properties. The scheme is a variant on the techniques of genetic programming and genetic algorithms, having components borrowed from each. By recognising the foundation of a quantum circuit as being a collection of gates, each operating on various categories of qubits and each taking parameters, the scheme can successfully search for most circuits. The algorithm is applied to the problem of entanglement production.

1 Introduction

At the rate of current technological advancement, computer memory and microprocessors are continually decreasing in physical size and increasing in speed - people of all backgrounds, be they scientists, engineers or home users, demand better performance from their computers every day. If these trends continue, by the year 2020 (Williams 1998) computer architectures will have become so small physically, that the effects of quantum mechanics will take over from the classical physics that today's computer scientists assume.

The study of quantum computers aims to understand what the capabilities and limitations of such systems could be if they were to be created. Now what makes quantum computers so exciting, is that quantum information theory envisions a world where quantum computers, as well as being able to handle most traditional classical algorithms, will be able to achieve far more than is possible with the famous Turing model - the basis of computer science today. Such seemingly wondrous abilities include: factoring large composite integers in polynomial time (assumed intractable by the popular public key encryption system RSA) (Shor 1994) and unbreakable quantum cryptography, where a secret key has been securely sent over a distance of 30km (Marand 1995).

So, it is clear that the study of quantum computation is well worthwhile, but still there have not been *that* many quantum algorithms discovered as yet. This is due to the fact that the generation of such algorithms or circuits is difficult for a human researcher: they are unintuitive and computationally intensive (see Section 2.1). Thus, it makes perfect sense to investigate the use of known automatic techniques, such as genetic programming and genetic algorithms which have proven to exhibit many desirable properties such as requiring no auxiliary information about the search space, except access to some kind of raw fitness function, and being highly robust.

Section 2 of this paper is a brief overview of the basics of quantum computing, and of the work done in the application of GP to quantum computing. Section 3 outlines the GP scheme for this paper, detailing its representation scheme and operators. Section 4 illustrates the aforementioned scheme applied to quantum entanglement production. Section 5 presents the results of section 4. Section 6 then discusses these results and the scheme in more detail.

2 Background

2.1 Quantum Computing

The basic processes in a quantum circuit run parallel to those of a classical electrical circuit, except for a few twists. First we start with the units of information: qubits. Like a classical bit, a quantum bit may take values in $\{0,1\}$ - called eigenstates. Unlike a bit, however, a qubit may also exist in a superposition of its eigenstates as long it remains unobserved. In this way, when we are describing the state of a qubit, we refer to its amplitudes as being the quantities that determine the weighting of the afore mentioned superposition. For example, consider the following two states written in the traditional 'ket' notation of quantum computing:

$$|\psi_1\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \quad |\psi_2\rangle = \tfrac{1}{2}|0\rangle + \tfrac{\sqrt{3}}{2}|0\rangle$$

Here, the amplitudes are $1/\sqrt{2}$, $1/\sqrt{2}$ and $1/2$, $\sqrt{3}/2$ respectively (note: amplitudes are complex). The probability of measuring a qubit as being in a particular eigenstate is simply the magnitude squared of the eigenstate's amplitude - observing a qubit in state $|\psi_1\rangle$ will result in $|0\rangle$ or a $|1\rangle$ with equal likelihood, and for $|\psi_2\rangle$ we'd get $|0\rangle$ 25% and $|1\rangle$ 75% of the time.

To describe the state of an n-qubit system, a total of 2^n amplitudes are needed; one for each of the 2^n possible eigenstates. Note also that the state of an *n*-qubit register is often represented as a 2^n by 1 column vector of amplitudes, and although one may always write the combined state of a number of qubits from their individual states, there are times where the converse is not possible. We illustrate these ideas with the following example. Consider the quantum state:

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \quad \begin{pmatrix} 1/\sqrt{2} \\ 0 \\ 0 \\ 1/\sqrt{2} \end{pmatrix}$$

If qubit 1 is measured (count as 0 from right) as a $|1\rangle$ (50% probability) then subsequent measurement of qubit 0 will result in $|1\rangle$ (100% probability). Similarly for measuring a $|0\rangle$. This effect is called quantum entanglement. This entangled pair is known as the EPR pair (Einstein, Podolsky, Rosen). Such a system may not be described as individual qubit state superpositions.

Due to the very nature of a quantum computer, its unobserved evolution through time is governed by the Schröedinger equation of quantum mechanics. From its solution it can be shown that this evolution is unitary. Without going into the inner workings of quantum computation, we note that this unitarity results in the ability to describe the operation of a quantum computer (without intermediate measurement) as a unitary matrix acting on the column vector for the initial register (so the matrix is 2^n by 2^n). Additionally, it has been shown that a quantum computer (an evolution of a qubit register) is equivalent to a sequence of quantum gates acting on qubits of the register. Each such gate has a corresponding unitary matrix, and there are various rules for combining the gate matrices to attain the overall circuit matrix. For example consider the EPR pair production circuit (*Fig.1*) operating on two qubits (an input of $|00\rangle$ results in the EPR pair from above).

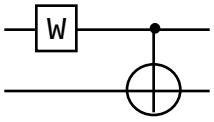

Figure 1: EPR pair production circuit

First the Walsh-Hadamard gate operates on qubit 1 (counting from 0 on the bottom) represented as the top horizontal line, and then a Controlled-NOT gate with qubit 1 as control and qubit 0 as target completes the circuit. Some other types of gates commonly used include conditional phase change, rotation, `NOT`, `NAND`, and some general gates which emulate all single or double qubit gates. We won't go into the workings of these gates here, but will note that gates require certain 'types' of qubits to operate on (ie. the Walsh-Hadamard operates on one qubit, the `CNOT` gate may have any number of controls/targets, etc.), and some require real valued parameters (ie. an angle θ for the rotation gate).

Finally, recall that the size of the column vector required to describe a register grows like $O(2^n)$ with the number of qubits, so it is clear that the complexity of a classical simulation of a quantum computer is exponential, making the simulation of large systems infeasible.

For further introduction into the background knowledge in quantum computing, see (Williams 1998).

2.2 Work Done So Far

Unlike other areas of application, there has been relatively little work done applying GP to quantum computation. Research has typically involved the investigation of basic representation schemes, and the use of these schemes in finding results competitive with hand calculation on the few known quantum algorithms. For example, three schemes outlined by Spector (Spector 1999), the traditional tree-based GP (Koza 1992) and a stack-based linear genome GP and stackless linear genome GP as alternatives, were applied to algorithms including the early promise problem (to determine whether a blackbox binary function on n-bits is uniform or balanced) and the and-or query problem (to find the result of a known boolean function over values returned by a blackbox function). In fact at the time, the solution obtained for the 2-bit and-or query problem, calling the blackbox function *only* once, was better than any previously known result. We shall return to the tree-based GP later, and analyse its appropriateness for problems in quantum computing (Section 6.3). The fitness evaluation was based on the probable outcome of measurement of the qubits after computation, in combination with misses and parsimony.

While the schemes put forward by Spector would typically be used to find circuits that met required outcomes for given inputs (ie. fitness cases), the scheme proposed by Williams (Williams 1999) uses the unitary matrix for a known quantum circuit to find possibly alternative circuits. In this way, more of the available information on a given algorithm is be utilised, but a circuit for the given problem must already be known. This is not necessarily bad, as it is very useful for finding alternate circuits, and could be used in conjunction with ideas of novelty (ie. penalising a circuit for looking similar to a known circuit) to produce such alternatives.

In addition, it should be noted that the above fitness functions can be obtained from each other: a complete unitary matrix can be constructed given enough fitness cases (ie. making the mappings of input registers as columns) and

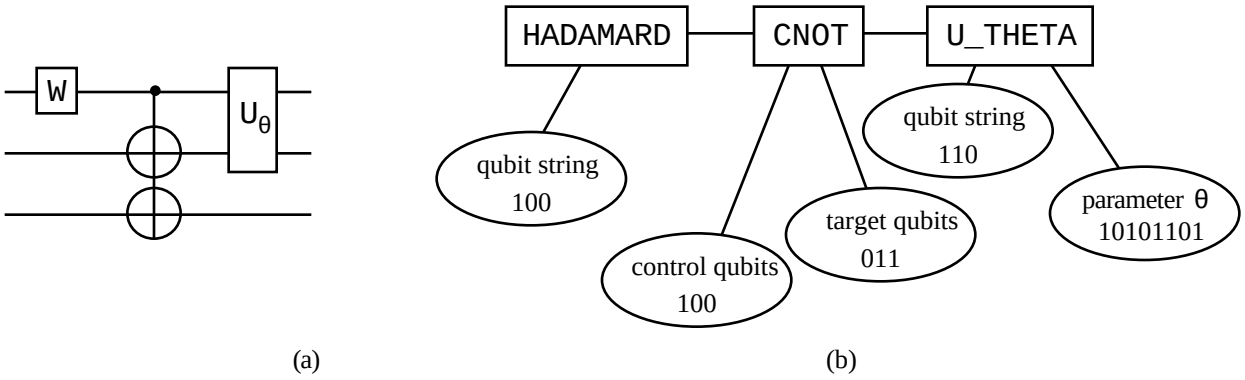

Figure 2: (a) a quantum circuit and its (b) corresponding representation

a partial matrix (where some entries are ignored) given a few fitness cases.

3 The GP Scheme

3.1 Representation

In section 2.1, we saw that a quantum circuit is a sequence of quantum gates, where the operation of each gate is entirely determined by:

- the gate's type
- the qubits acted on by the gate
- the values of any parameters required by the gate

In addition we noted that the second and third items of the above list, qubit operands and parameters, appear in a variety of flavours. For example, a Controlled-NOT gate takes some number of control qubits, from which it may NOT (ie. swap) the state amplitudes of its target qubits. The generalised 2-qubit gate takes four real parameters (ie. different kinds of rotations, etc.). So we see in general, that a gate has a type, a number of sets of qubit operands in different categories and some sets of parameters in different categories. This is the basis for the GP scheme.

Consider a (quantum) circuit. It is represented as a list of gate structures, where the size of this circuit (number of gates) is variable, as in the variable length GA. Each gate structure contains its type, which is one of an allowable set of gate types including the usual `CNOT`, `HADAMARD` and other unitary evolutions, but *also* an `OBSERVE` gate which can measure certain qubits in the register, thereby collapsing superposition (a vital technique in several famous algorithms), and a binary string for each category of qubit and parameter for that gate.

The length of each qubit bit-string is simply equal to the number of qubits n_qubits in the system. The i^{th} bit in a qubit binary string determines whether or not qubit $(n\_qubits - i)$ of the register belongs to the corresponding category of qubits for that gate. In other words, if the last bit of a binary string is *1* then qubit 0 is acted on, if the first bit were *0* then the last qubit $(n\_qubits - i)$ is not acted on. As we cannot have a qubit belonging to more than one category, we have a hierarchical chain of precedence, where each category is ranked uniquely. In this paper, we have at most two qubit categories (targets and controls), the target category has been arbitrarily chosen to take precedence over the control qubits.

The binary string for a parameter of a gate decodes to a real decimal value, by simply mapping it to a specific interval. In this paper, we only have angles of rotation, so this interval is simply $[-\pi, \pi]$. The length for parameter strings is arbitrary chosen, so that the resolution of the mapping is as fine as desired (ie. more bits give closer approximations to a continuous number line). This representation scheme is illustrated with the following (*Fig.2*):

A gate which is not syntactically correct, which does not act on *any* qubits, or has no control or target qubits when it is required to, is treated simply as the identity transformation.

3.2 Genetic Operators

Making up the set of operators, we have crossover and mutation. Crossover operates on all the levels of an individual's structure: the gates, each category of qubit operands and each parameter type. Gate crossover between two parent computers (which may have different numbers of gates) consists of picking a gate from each parent at random, and then swapping all gates between the parents after these two points - the canonical variable length GA crossover, closely related to tree-based GP crossover. Crossover between qubit and parameter binary strings can only occur between strings of like category, and proceeds in the same way as the crossover operator for the fixed length GA: pick a crossing point, and then swap bit values between the two strings after the point. This crossover between like structures is analogous to the crossover of tree-based GP with automatically defined functions (ADFs) and result producing branches (RPBs). By restricting crossover of an ADF with another ADF and an RPB with another RPB, essential syntactical correctness is preserved and the RPBs and ADFs of individuals evolve in parallel (which is a good

thing, as their roles are quite different; 'ideas' or schemata that are good for ADFs may not be as good within an RPB).

Two methods for combining these crossover operators have been investigated: *uniform* and *weighted* crossover. Uniform crossover on two randomly paired parents, consists of first picking a gate, or qubit operand set or parameter completely at random from anywhere within the computer (ie. list of gates), where each gate, set of operands and parameter has equal probability of selection. Then we pick a similar structure (either gate, qubit operand set of like category or like parameter) from the second parent, at random. Finally, the appropriate crossover operator is applied to the selected gates/binary strings. In this way, we are thinking of a computer as a tree with each gate, qubit operand set and parameter as nodes, picking a node at random from parent 1, and then a similar node from parent 2. Weighted crossover proceeds by simply picking an operator at random, where the operators (gate, control/normal qubit, target qubit, parameter) have associated weights. Once the operator is decided upon, two appropriate structures within the parents are selected randomly, and operated on.

Mutation is performed with small probability (typically 0.001, as mutation is more of an insurance against loss of important schemata, than a search procedure), where a gate is mutated by replacing it with a new random gate.

3.3 The Fitness Measure and Fitness Cases

The fitness function employed for this scheme is standardised, that is all fitness values lie in the range $[0,1]$ with smaller values being better. Firstly, given a set of fitness cases consisting of input registers and desired output registers, we run the set of input registers through a computer and define its error as...

$$error = \sum_i \sum_j \left| o_{ij} - d_{ij} \right|$$

...where we take for each case i, the sum of the magnitudes of the differences between the amplitudes of the outcome registers o_{i*} and the corresponding desired output registers d_{i*} (j over the number of states), and take the sum of all of these. In this way, an error of zero is clearly optimal, to adjust these errors to fitness's within the required interval, we divide through by the error of the worst individual so far. In this way, at any one time, all errors *must* be equal to or less than this value, and so dividing through by the worst error will map the errors to the desired range. Of course, in any post-run analysis, one would have to go back, and re-scale for any changes in worst-so-far through the run, so that progress remains objective.

3.4 Summary and Steps for the GP

As a whole, the scheme proceeds in the following fashion:

(1) the initial population of computers is created randomly, where the size of the population is M, the maximum number of gates in a computer is max_gates_{init} and the maximum number of qubits operated on by a gate is max_qubits_{init}.
(2) the fitness of each individual is evaluated with the provided fitness cases, relevant to the problem.
(3) a new population is selected, using *roulette wheel* selection, each slice of the wheel is simply proportional to $(1 - fitness)$
(4) the new individuals are paired off randomly, and are crossed with probability p_c, according to the uniform or weighted crossover schemes
(5) mutation is performed with low probability pm, in an attempt to retain/regain useful schemata
(6) repeat steps 2-5 until either G generations have been run, or some threshold for fitness has been achieved.

Now that we have laid the foundations for quantum computing, and have put forward a plausible scheme for evolving quantum circuits, we are equipped to tackle some problems with the adaptive plan (section 4 and 5) and further discuss the reasons and features of the scheme, and any observations from its application (section 6).

4 Applications

4.1 Quantum Entanglement Production

Recall the example of section 2.1 (*Fig.1*) of a circuit that produces the important EPR entangled pair from a classical input of $|00\rangle$. As we have already mentioned, entanglement is a very important feature of quantum mechanics which is partially responsible for the power of many of today's quantum algorithms. To test the GP scheme detailed above, it was applied to the problem of generating circuits for production of 2, 3, 4 and 5 maximally entangled qubits respectively (*Table.1*)

The parameter length and the (overly) extensive gate set were used to test the limits of the scheme, as it turns out the population size (restricted by memory considerations) was easily sufficient for the problem. Only one fitness case was used, because this was the necessary function of the circuit (outputs from other inputs are irrelevant, we just wanted an easy method of producing entangled sets of qubits). The crossover and mutation probabilities were chosen intuitively, as was the maximum number of gates (considering the simplicity of the EPR circuit of *Fig.1*).

OBJECTIVE:	Find a quantum circuit that produces n maximally entangled qubits in the form $1/\sqrt{2}(\|00...0\rangle + \|11...1\rangle)$. ie. generalised EPR pair
QUANTUM TERMINALS:	number of qubits in register $= n = \{2,3,4,5\}$, $param\_length = 8$
QUANTUM GATES:	`IDENTITY, HADAMARD, U_THETA, CNOT, CPHASE, NOT, NAND, OBSERVE`
FITNESS CASES:	One case: input register $\|00...0\rangle$ output register $1/\sqrt{2}(\|00...0\rangle + \|11...1\rangle)$
RAW FITNESS:	Sum of the distances between the amplitudes of the desired output register and the actual output register
STANDARDISED FITNESS:	the raw fitness divided by the worst so far raw fitness
PARAMETERS:	$M = 5000$, $G = 50$, $f_t = 0.001$, $max\_gates = 3$, $max\_qubits = 2$, $p_c = 0.8$, $p_m = 0.01$, uniform crossover used
TERMINATION PREDICATE:	Max. generations G exceeded, or threshold fitness f_t achieved

Table.1: Tableau for entangled qubit production problem

5 Results

5.1 Quantum Entanglement Production

For each of the chosen sizes of the problem, a completely correct solution was discovered by the GP scheme. Due to the apparent simplicity of the solutions (with hind-sight) and the large population size, most of the solutions were generated in between 2 and 8 generations (where each generation took between about 30 seconds and several minutes, due to the complexity of quantum computing).

For $n = 2$, the circuit found had no redundancy, and was identical to the one shown in *Fig.1* above. For $n = 3$ two revealing solutions were found (*Fig.3*)

```
Controlled NOT gate:  control qubits target qubits 0, 1, 2
Controlled NOT gate:  control qubit 2 ; target qubits 0
Hadamard gate:  qubit 1
Controlled NOT gate:  control qubits target qubits 0, 2
Controlled NOT gate:  control qubit 1 ; target qubits 0
Controlled NOT gate:  control qubit 1 ; target qubit 2

Controlled NOT gate:  control qubits target qubits 1, 2
Controlled NOT gate:  control qubit 1 ; target qubits 0
Controlled NOT gate:  control qubit 0 ; target qubits 1
Hadamard gate:  qubit 2
Controlled NOT gate:  control qubit 1 ; target qubit 0
Controlled NOT gate:  control qubit 1 ; target qubits 0
Controlled NOT gate:  control qubit 2 ; target qubits 0
Controlled NOT gate:  control qubit 0 ; target qubit 1
```

Figure.3: two solutions for 3-entangled qubit problem

The circuit diagrams for these two solutions (with redundancies removed by hand):

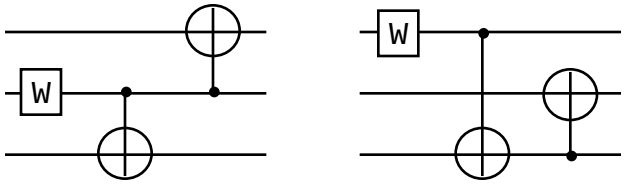

Figure.4: circuits for the 3-entangled qubit problem (left corresponds to the top listing of the circuits in Fig.3

And finally, we have the structures and circuits obtained for $n = 4$ and $n = 5$ below:

```
Hadamard gate:  qubit 2
Controlled NOT gate: control qubit 2 ; target qubits 0, 1, 3

Hadamard gate:  qubit 2
Controlled NOT gate: control qubit 2 ; target qubits 0,1,3,4
```

Figure.5: solutions for the 4 and 5-entangled qubit problems

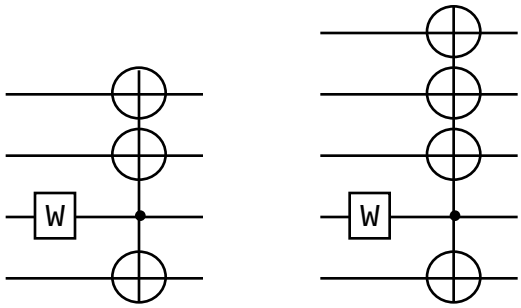

Figure.6: circuits for the 4 and 5-enttangled qubit problem (left corresponds to the top listing of the circuits in Fig.5

6 Discussion

6.1 Entanglement Production

The application of the proposed GP scheme to entanglement production circuits was successful - a number of circuits for different sized registers were found that produced maximally entangled registers of qubits. However, it is clear that the methods for attaining these results could have been improved in a number of ways. For example, it would have been prudent to remove non-essential gates from the gate-set, leaving only the Walsh-Hadamard and the Controlled-NOT gates. This *would* have increased the speed of the calculations. In fact, a small number of runs *were* conducted this way, and sometimes a result was actually produced in the initial population, which clearly would have been a good representation of the space. However the purpose was also to test the presented scheme, which it did.

In fact, already from the relatively small number of circuits we have produced for this problem, we can see the general solution: the Walsh-Hadamard gate takes a qubit to a balanced superposed state, after which this superposition is spread to the other qubits by using it as control for Controlled-NOT gates which then target the other qubits (see *Fig.4* and *6*). In addition, this spread of entanglement can be achieved by taking a qubit already targeted previously, as a control and targeting a new qubit. In summary, *a* general solution has been obtained, and we have gained a better understanding of the generation solution.

6.2 Further Discussion and Analysis of the Scheme

Consider now the presented GP scheme in its most general case, where not only are there two qubit operand categories and one parameter type, but we have c operand categories and p parameter types, with the number of qubits in the system equal to q, the number of bits used to represent a parameter equal to k, and the number of gate types equal to t. The total number of ways in which we may operate on all the qubits in any category (ie. simply decide which category a qubit is in) is c^q. To operate on all but one of the qubits using any category, we have...

$$\binom{q}{q-1} c^{q-1}$$

...and so on. Hence the number of ways in which the qubit bit strings for a gate may be configured, using all categories, is simply the sum of these possibilities:

$$\sum_{i=0}^{i=q} \binom{q}{i} c^i$$

Then in a similar fashion, if we choose to use all but one of the categories, then we have the number of ways as:

$$\binom{c}{c-1} \sum_{i=0}^{i=q} \binom{q}{i} (c-1)^i$$

And so the total number of ways in which we may operate on the system of qubits is:

$$\sum_{j=0}^{j=c} \binom{c}{j} \sum_{i=0}^{i=q} \binom{q}{i}^j$$

Now consider the parameters for some gate. The number of values taken by a single known parameter is just 2^k. So the number of ways we may choose an unknown parameter is

$$\binom{p}{1} 2^k$$

And following from this, the number of ways in which we may choose two parameters for a gate is:

$$\binom{p}{2} 2^{2k}$$

So the total number of ways we may choose the parameter settings for a known gate is the sum:

$$\sum_{h=0}^{p} \binom{p}{h} 2^{hk}$$

Now, if the gate is of unknown type, then the number of possible gate configurations is:

$$\left(\sum_{h=0}^{p} \binom{p}{h} 2^{hk} \right) \left(\sum_{j=0}^{j=c} \binom{c}{j} \sum_{i=0}^{i=q} \binom{q}{i}^j \right)^t$$

So, if we can have exactly n gates in a computer, then the total number of possible computers is simply:

$$\left[\left(\sum_{h=0}^{p} \binom{p}{h} 2^{hk} \right) \left(\sum_{j=0}^{j=c} \binom{c}{j} \sum_{i=0}^{i=q} \binom{q}{i}^j \right)^t \right]^n$$

And finally, if we are allowed to have a total number of n gates, then the solution space is exactly:

$$\sum_{g=0}^{g=n} \left[\left(\sum_{h=0}^{p} \binom{p}{h} 2^{hk} \right) \left(\sum_{j=0}^{j=c} \binom{c}{j} \sum_{i=0}^{i=q} \binom{q}{i}^j \right)^t \right]^g$$

The number of schemata follows from this result, by noting that: the total number of ways in which we may

operate on all the qubits in any category (ie. simply decide which category a qubit is in) is simply $(c+1)^q$ instead of c^q (since we are essentially including the meta-don't-care-symbol), and similarly the number of values taken by a single known parameter becomes 3^k from 2^k. This gives the number of schema for a computer of length up to and including n as:

$$\sum_{g=0}^{g=n} \left(\sum_{h=0}^{p} \binom{p}{h} 3^{hk} \left(\sum_{j=0}^{j=c} \binom{c}{j} \sum_{i=0}^{i=q} \binom{q}{i} (j+1)^i \right)^t \right)^g$$

In addition, any one of these computers may be generated with the operators we have defined, simply from the fact that we have a mutation operator, which generates completely random gates. Thus *even* if crossover prevented us from getting certain computers (which it doesn't, it simply recombines ideas, losing no information), mutation alone, although improbable, is able to generate any such computer eventually.

6.3 Comparisons and Alternative Schemes

Two other schemes were considered during the course of investigations, as well. The first was similar to the one proposed here, in that a computer was a sequence of gates, each gate could act on any qubit and could have parameters. The idea is for the gates to be represented as a *variable width* matrix, of height equal to the number of qubits. The matrix is composed of gate structures, where a gate in row i operates on qubit i of the qubit register (and of course, required parameters would be contained within the gate structure). Those gates that operate on more than one category of qubit (ie. control and target), require another version of the same gate type to appear in the same column, where *its* row determines the qubit *it* operates on. In this way, if a CNOT control-qubit gate is found at (a,b) and a CNOT target-qubit gate is found at (c,b) then a CNOT control qubit a, target c becomes part of the circuit at the point b along the circuit. This scheme did show some promise, however the complex crossover operators required prevented its investigation.

Traditional tree-based GP, as outlined in (Spector 1999) was the other scheme considered. In Spector's work with the TBGP, one clear advantage and disadvantage become apparent. Due to the functional nature of the tree implementation, it is relatively easy to enable evolved quantum circuits to be scalable, or at least make it easier. That is, an evolved circuit not only works on n qubits but also on $n+1$ qubits (in theory). This is because instead of only having permutable constants for the qubit operands, parameters, etc. there can also be constants such as the number of qubits in the system - as all of these real/complex/integral values are implemented as the terminal set. So to evolve a scalable circuit, one would test each individual with a handful of values for the number of qubits terminal. The disadvantage, as with any tree-based GP and most if not all such evolutionary techniques, is redundancy, but more importantly there is also a separation of return value and side-effect on the qubit register, making it particularly hard for GP to retain an important return value and modify a side effect. In other words, arithmetic functions that operated solely on values returned by other functions/gates and terminals were mixed in with gates that actually affected the register.

Unlike the TBGP, the variation GP scheme presented here does not have this side-effect/function value return problem, as the qubit operands and parameters are completely separated from the gates (in terms of crossover, etc). Furthermore, schemata that may be important in one category of qubit operands or parameter may not be important in other categories, and so our scheme makes sure that good schemata within one category have the opportunity to develop. For example (see *Fig.6*) it is clear from the discussion above, that once a Walsh-Hadamard had operated on a qubit, it was unproductive to have target qubits for Controlled-NOTs on the same qubit, only controls - schemata for controls at this location should have representatives that are highly fit. As far as scalability goes, we demonstrated that it is possible, *even* by hand, to deduce information about the general scalable solution to a problem.

Although non-evolutionary heuristics such as simulated annealing are highly successful in many application areas, they may run into problems here, due to the irregular fitness landscape with many local minima and maxima. Evolutionary schemes, on the other hand, are particularly well suited to handling this type of problem, with their typically robust, parallel methods of search.

7 Conclusion

We have presented a successful GP scheme for evolving quantum circuits for problems of given behaviour. In doing so, we have outlined the importance of recognising the inherent structure in quantum circuits, thus enabling the development of operators that can both generate any quantum computer (over the given gate set), and focus on parts of the computer that may be prone to similarities.

By generating a relatively small number of quantum circuits for maximal entanglement production, we have learnt much about the general form for an arbitrarily sized entanglement production circuit. Thus, we have recognised the power of GP, even though only small systems were used due to the inherent complexity of quantum computation.

8 Future Work

Although a preliminary schemata analysis has been made here, a more complete investigation would surely shed light

on our adaptive scheme, possibly further justifying the choice of operators or suggesting alterations. Although we did apply the scheme to a real-world quantum computing problem it was small in gate number and type. It would be interesting to see how it fares against harder problems such as teleportation, Grover's algorithm, etc. Cluster computing could be employed to deal with the problems in complexity (for large problem sizes). Finally, with the framework laid by the OBSERVE gate, the implementation of automatically defined gates, iteration, recursion, etc could be investigated, as these ideas have proved invaluable in many other GP applications.

Acknowledgments

I'd like to thank Dr. C. P. Williams (of JPL and Stanford University) for suggesting a number of quantum computing problems that lend themselves to solution by genetic programming.

Bibliography

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

Marand, C. and Townsend, P. Quantum Key Distribution Over Distances as Long as 30km. In *Optics Letters*, Vol. 20, No. 16, 15 August (1995), pp. 1695-1697.

Shor, P. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science* (1994), pp. 124-134.

Spector, L., Barnum, H., Bernstein, H. J. and Swamy N. Quantum Computing Applications of Genetic Programming. In *Advances in Genetic Programming*, Vol. 3, (1999), pp. 135-160.

Williams, C. P. and Clearwarter, S. H. 1998 *Explorations in Quantum Computing*. Springer-Verlag New York, Inc.

Williams, C. P. and Gray, A. Automated Design of Quantum Circuits. In *Lecture Notes in Computer Science* Volume 1509 - issue dedicated to "Quantum Computing and Quantum Communications", C. P. Williams (ed.) (1999), Springer-Verlag .