# Finding a Better-than-Classical Quantum AND/OR Algorithm using Genetic Programming

**Lee Spector**
School of Cognitive Science
Hampshire College
Amherst, MA 01002, USA
lspector@hampshire.edu

**Howard Barnum**
School of Natural Science
Institute for Science and Interdisciplinary Studies (ISIS)
Hampshire College
Amherst, MA 01002, USA
hbarnum@hampshire.edu

**Herbert J. Bernstein**
School of Natural Science
Institute for Science and Interdisciplinary Studies (ISIS)
Hampshire College
Amherst, MA 01002, USA
hbernstein@hampshire.edu

**Nikhil Swamy**
Box 1026
Hampshire College
Amherst, MA 01002, USA
nikhil_swamy@hampshire.edu

**Abstract- This paper documents the discovery of a new, better-than-classical quantum algorithm for the depth-two AND/OR tree problem. We describe the genetic programming system that was constructed specifically for this work, the quantum computer simulator that is used to evaluate the fitness of evolving quantum algorithms, and the newly discovered algorithm.**

## 1 Introduction

Quantum computers use the dynamics of atomic-scale objects to store and manipulate information. The behavior of atomic-scale objects is governed by quantum mechanics rather than by classical physics, and the quantum mechanical properties of these systems can be harnessed to compute certain functions more efficiently than is possible on any classical computer [1]. For example, Shor's quantum factoring algorithm finds the prime factors of an $n$-digit number in time $O(n^2 \log(n) \log\log(n))$ [2], while the best known classical factoring algorithms require time $O(2^{n^{\frac{1}{3}} \log(n)^{\frac{2}{3}}})$ and many researchers doubt the existence of polynomial-time classical factoring algorithms [3, 4]. An unambiguous case of the superiority of quantum computation was provided by Grover, who showed how a quantum computer could find an item in an unsorted list of $n$ items in $O(\sqrt{n})$ steps, while classical algorithms require $O(n)$ [5].

Only a few, small-scale quantum computers have been built to date, and it is not yet clear how larger, more practical quantum computers will be constructed, or if it will even be possible to do so [6]. Nonetheless, the predicted efficiencies of quantum computing are so significant that the study of quantum algorithms has attracted widespread interest. For those new to quantum computing Steane provides a good overview [7], Braunstein provides an accessible on-line tutorial [8], Williams and Clearwater provide a text with a CD ROM containing Mathematica code for simulating quantum computers [9], and Milburn provides an introduction for the general reader [10].

The widespread application of quantum computation presupposes the existence of both quantum computer hardware and quantum software that solves practical problems. But the development of quantum algorithms is not trivial. Williams and Clearwater 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. [9, p. 42]

One approach to this problem is to use automatic programming techniques such as genetic programming to automatically generate new quantum algorithms. The resulting algorithms may be useful in their own right and they may also provide new insights to human quantum programmers.

Genetic programming is an automatic programming technique that evolves programs by mimicking natural selection [11, 12, 13, 14, 15]. In the simplest version, a set of programs undergoes processes such as "mutation" (small random changes) and "crossover" (exchanging a segment of one program with a segment of another). Selection for desired properties—such as computing a certain function with low error probability, doing it quickly, with the smallest number of calls to a particular subroutine, or with a minimum amount of code—is then applied, by running the programs on a sample of the possible input data (called *fitness cases*), and scoring them on how close they are to the desired characteristics. Higher-scoring programs participate with higher probability in the process of reproduction involving crossover, etc., and the procedure is iterated on the resulting new generation of programs.

In this paper we describe work on the production of quantum algorithms by genetic programming. We start with an overview of our methods, including a description of a new genetic programming system that we have developed specifically for this work. We then present a brief synopsis of prior results and describe a new, evolved better-than-classical quantum algorithm for the depth-two AND/OR tree problem. We conclude with a discussion of the potential significance of efficient algorithms for AND/OR problems.

## 2 Methods

### 2.1 Quantum Computer Simulation

Because practical quantum computer hardware is not yet available, we must test the fitness of evolved quantum algorithms using a quantum computer simulator that runs on conventional computer hardware. This entails an exponential slowdown, so we must be content to simulate relatively small systems.

The basic unit of information in a quantum computer is a *qubit*, which is like a classical *bit* except that a qubit can exist in a superposition of states and can become *entangled* with other qubits—that is, its value can be linked to the values of other qubits in non-classical ways.

Our simulator follows the common practice of representing the state of the an $n$-qubit quantum computer as a vector of $2^n$ complex numbers, for which we use the notation $[\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{2^n-1}]$. Each of these complex numbers is a *probability amplitude* corresponding to one of the $2^n$ classical states (often called the *computational basis states*) of an $n$-bit classical computer. Following the tradition in the quantum computation literature we label the computational basis states using "ket vector" notation, as $|b_{n-1}b_{n-2}\ldots b_j \ldots b_0\rangle$, where each $b_j$ is either 0 or 1. The state labels can be abbreviated by interpreting them as binary numerals and changing to decimal notation; that is, we can write $|k\rangle$ in place of $|b_{n-1}b_{n-2}\ldots b_j \ldots b_0\rangle$ where $k = b_0 + 2b_1 + 4b_2 + \ldots + 2^{n-1}b_{n-1}$. For example we can write $|14\rangle$ in place of $|1110\rangle$.

Our quantum algorithms always begin in the computational basis state $|0\rangle$. When the state of quantum computer is read (*measured*) in the computational basis at the end of a computation, the probability of finding it in the computational basis state $|k\rangle$ is $|\alpha_k|^2$. If one is interested in the probability of finding some subset of the system's qubits to have a particular pattern then one can sum the squared moduli of the amplitudes for all states with labels containing the pattern. For example, the probability of reading the right-most 2 qubits of a 4-qubit quantum computer as "01" (that is, the probability of reading a state with label $|??01\rangle$) is the sum of the squared moduli of the amplitudes for $|0001\rangle$, $|0101\rangle$, $|1001\rangle$, and $|1101\rangle$, or $|\alpha_1|^2 + |\alpha_5|^2 + |\alpha_9|^2 + |\alpha_{13}|^2$.

Computations on the simulated quantum computer are modeled as sequences of linear transformations applied to the vector of probability amplitudes. These transformations, which are often called *quantum gates* and which can be represented as matrices, correspond to the physical manipulations possible on quantum mechanical systems. Some of these transformations act like classical logic gates, moving probability from one computational basis state to another, while others produce the non-classical effects of *superposition* and *interference* that are responsible for the unique efficiencies of quantum computation. Over the course of our work we have used several different sets of quantum gates. Our current system uses the following gates which can be presented in simple matrix form, along with ORACLE and MEASUREMENT gates that are described below. Parameters listed in square brackets ("[]") are qubit references with higher-order qubits listed first, and parameters listed in parentheses ("()") are real numbers:

- The HADAMARD gate, $H[q]$:

$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

- Simple rotation, $U_\theta[q](\theta)$:

$$\begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

- Generalized rotation, $U2[q](\alpha, \theta, \phi, \psi)$:

$$U_2 = \begin{bmatrix} e^{-i\phi} & 0 \\ 0 & e^{i\phi} \end{bmatrix} \times \begin{bmatrix} \cos(\theta) & \sin(-\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$\times \begin{bmatrix} e^{-i\psi} & 0 \\ 0 & e^{i\psi} \end{bmatrix} \times \begin{bmatrix} e^{i\alpha} & 0 \\ 0 & e^{i\alpha} \end{bmatrix}$$

- Controlled NOT, $CNOT[control, target]$:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

- Conditional phase, $CPHASE[control, target](\alpha)$[1]:

$$CPHASE = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\alpha} \end{bmatrix}$$

Note that a matrix must be of size $m \times m$ to be applied directly to an $m$-element column vector, and that the above matrices can therefore not in general be directly applied to the $2^n$-element amplitude vectors that characterize the states of an $n$-qubit quantum computer. A detailed description of how these gates are applied to multi-qubit systems can be found in [17].

[1] Note that this is a different form of CPHASE than was used in our previous work [16, 17].

An ORACLE gate is a permutation matrix that computes a Boolean function that may change from instance to instance of a given problem. This is useful because many existing quantum algorithms solve tasks that involve determining properties of unknown "black box" functions. For example, we might be given a function of $m$ binary inputs and wish to know what combination of inputs causes the function to output "1". This, with the proviso that exactly one input combination will produce an output of "1", is the database search problem solved by Grover's quantum algorithm. The ORACLE gate implements this sort of "black box" function, and a satisfactory algorithm must produce the proper outputs for all appropriate instances of the ORACLE.

We implement an ORACLE gate for an $m$-input Boolean function as an $(m+1)$-qubit gate that inverts its output qubit wherever the Boolean function of the input qubits yields "1". The output qubit is inverted by swapping the amplitudes between all states differing only with respect to the output bit. For example the OR function of two inputs, which outputs "1" for all combinations of inputs except "00", is implemented as follows:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The latest revision of our quantum computer simulator also supports the simulation of intermediate one-qubit MEASUREMENT gates for use in decision problems (that is, problems with yes/no answers). The measurement gates disrupt the state of the simulated quantum computer, just as intermediate physical measurements necessarily disrupt the state of real quantum computers, but they may also yield useful information that can be used to improve the overall efficiency of certain quantum algorithms by decreasing the expected number of oracle calls that must be made before measuring the correct answer with sufficient probability.

We implement one-qubit measurement gates by making the following additions to our quantum computer simulator:

- We add an oracle-call counter OC and a measurement database MDB.
- Whenever the oracle is called, we increment OC.
- We add a MEASURE-0 gate of one argument Q which adds (0, P(Q,0), OC) to MDB, where P(Q,0) is the probability of reading qubit Q as 0 at the time MEASURE-0 is called. MEASURE-0 also sets the amplitudes of all states with Q = 0 to 0.
- We add a MEASURE-1 gate of one argument Q which adds (1, P(Q,1), OC) to MDB, where P(Q,1) is the probability of reading Q as 1 at the time MEASURE-1 is called. MEASURE-1 then sets the amplitudes of all states with Q = 1 to 0.
- We ensure that every simulated quantum algorithm ends with a MEASURE-0 immediately followed by a MEASURE-1 on the same qubit.
- We add an ERROR-PROBABILITY function of one argument ANSWER which should be 0 or 1. (Recall that this implementation only supports intermediate measurements for decision problems.) If ANSWER is 0 then ERROR-PROBABILITY returns the sum of the P values for all entries in MDB that start with 1. If ANSWER is 1 it returns the sum of the P values for all entries in MDB that start with 0.
- We add an EXPECTED-QUERIES function of no arguments that returns the sum of (P value $\times$ OC value) for all elements in MDB.

MEASURE-0 and MEASURE-1 are new quantum gate functions that can be called within quantum algorithms. ERROR-PROBABILITY and EXPECTED-QUERIES are new utility functions to be called at the end of a simulated run to determine the error probability and the expected number of oracle queries. ERROR-PROBABILITY and EXPECTED-QUERIES are to be used in place of the standard measurement function whenever intermediate measurements are included in the simulated quantum algorithm.

## 2.2 Linear Genome Steady-State Genetic Programming

In previous work we described the evolution of quantum algorithms by means of standard, tree-based genetic programming, by means of stack-based linear genome genetic programming, and by means of stackless linear genome genetic programming [16, 17]. We started with the most conventional genetic programming framework and modified our approach incrementally in response to perceived demands of the quantum computing application area. While we do not have strong statistical evidence in favor of the modifications (which would be expensive to obtain because of the high cost of the many embedded simulations), the continual improvement of our results gives us confidence that our methods have become better adapted to the quantum computing domain. Our latest work is based on a re-implemented stackless linear genome genetic programming system. In contrast to our previous system the new system is *steady-state* rather than generational, it supports true variable-length genomes and lexicographic fitness comparisons [18], and it is well-suited to hybrid genetic/local-search methods and to distributed operation across a cluster of workstations.

An individual in the system is a pair consisting of a fitness value and a quantum algorithm. The quantum algorithm is a list of quantum gates, and the fitness value is either the special symbol NO-COMPUTED-FITNESS (indicating a new individual that has not yet been evaluated) or a list of fitness component values. Fitness components currently implemented include MISSES, EXPECTED-QUERIES, MAX-

ERROR, and NUM-GATES. For all fitness components lower numbers are interpreted as being better. MISSES is the number of fitness cases on which the algorithm performs unacceptably,[2] EXPECTED-QUERIES is the number of oracle queries that one expects to make to get the answer (see above), averaged over all fitness cases,[3] MAX-ERROR is the maximum, over fitness cases, of the probability of getting the wrong answer for the case, and NUM-GATES is the number of quantum gates in the algorithm. Fitness comparisons are lexicographic, meaning that individuals with better values for the first component will be judged as better than individuals with worse values for the first component, regardless of the values of the other fitness components. When the values of the first components are equivalent the comparison depends on the values of the second components, and so on. One can optionally set a threshold below which differences between fitness component values are ignored; this can help to avoid preference for algorithms that appear better only because of minuscule round-off errors. In the run that produced the new AND/OR algorithm presented below, we used fitness values of the form: ($max$(EXPECTED-QUERIES, 1), MISSES, MAX-ERROR, NUM-GATES).

Our system begins execution by creating population of (usually 1000) individuals with random quantum algorithms. In each time-step our system chooses a random *genetic operator* and executes it, possibly changing one individual in the population. Most operators work with copies of individuals in the population, selected by choosing a small number (3 for the run producing the algorithm described below) of individuals and conducting a *selection tournament.* If one or more individuals in the selection tournament group has NO-COMPUTED-FITNESS then one of these (chosen randomly) will be designated as the winner and will be returned from the selection process; otherwise the individual with the best lexicographic fitness will be returned. Each execution of a genetic operator produces a possibly new individual which the system will then attempt to insert back into the population. The system does this by choosing a random individual in the population and conducting a two-individual *replacement tournament.* In a replacement tournament any individual with NO-COMPUTED-FITNESS is first evaluated for fitness. The individual with better lexicographic fitness is retained in the population, and the other individual is discarded. We also provide a system parameter PERCENT-LOSERS-WIN that allows one to specify that some small percentage of the time (10% in the run that produced the new algorithm described in Section 4) the individual with worse fitness will be retained and the individual with the better fitness will be discarded; this can sometimes help to maintain diversity in the population.[4] The system maintains a record of the best individual that it has seen throughout the run, and reports this individual and its fitness upon termination. The run terminates when a pre-specified fitness value is achieved, or when the user manually triggers termination.

The following are the genetic operators currently included in the system:

- REPRODUCTION: Produces a new copy of an individual selected from the population, unchanged.
- CROSSOVER: Produces a new individual constructed from a random initial segment of one individual appended to a random tail segment from a possibly different individual.
- MUTATION: Produces a new individual by substituting one randomly generated instruction for one instruction in an individual selected from the population.
- INSERTION: Produces a new individual by sandwiching a random middle segment of one individual between random initial and tail segments of a possibly different individual. The initial and tail segments might overlap or fail to include all instructions from the 'outer' individual.
- MUTANT-INSERTION: Produces a new individual by inserting a new random program within the program of an individual selected from the population.
- DELETION: Produces a new individual by removing some middle segment of an individual selected from the population.
- ANGLE-MUTATION: Produces a new individual by selecting an individual from the population and one gate in that individual, and by randomly changing one of the angles in that gate (if it contains an angle).
- MINIMIZATION: Produces a new individual by removing gates from an individual selected from the population. Each gate is examined for removal, starting from the beginning, and the removal is accepted if it results in an equal or better lexicographic fitness. Note that this is an expensive operator, as it requires as many fitness tests as there are gates in the selected algorithm.
- PAIR-MINIMIZATION: Like MINIMIZATION but examines every *pair* of gates for possible removal. This is even more expensive, requiring O($n^2$) fitness tests where $n$ is the number of gates in the selected algorithm.
- MULTIPLE-ANGLE-PERTURBATION: Produces a new individual by selecting an individual from the population and adding small constants to some small number of its angles (randomly chosen).

Note that the minimization operators perform a sort of "lo-

[2]We say that an algorithm's performance on a particular case is "acceptable" for the sake of MISSES if its probability of producing the correct answer is at least 0.52. This is far enough from 0.5 to be confident that it is not due to a simulator round-off error. MISSES provides only a weak test of "acceptability," and we rely on the MAX-ERROR fitness component to encourage the evolution of low-error algorithms.

[3]In many cases one will want to use the maximum rather than the average here; for the results described in this paper this is not material.

[4]We also stipulate that when a new individual, created by a genetic operator, has the best fitness seen so far in the current run, then it will always win the replacement tournament.

cal" hill-climbing search that is integrated into the genetic search process. A natural extension is to add additional operators that perform more intensive local search (optimization) procedures, for example by systematically varying all of the angles in the gates of the selected quantum algorithm.

Another natural extension allows for effortless distribution of runs across a cluster of workstations. One must only add the following additional operators:

- EMIGRATION: Selects an individual from the population and sends a copy of that individual to a different, randomly chosen workstation. The receiving workstation stores the individual in a queue for possible use by an IMMIGRATION operator.
- IMMIGRATION: If the immigration queue is empty then this does nothing. Otherwise it returns the first individual in the queue for possible insertion into the population (via replacement tournament).

A distributed run is conducted by starting an independent run on each workstation and by informing each copy of the system of the available EMIGRATION targets. Because there is no global coordination one must examine the results on all workstations and select the best result as the output from the distributed computation.

## 3 Prior Results

We reported previous results in [16, 17]. By sometimes evolving not just quantum algorithms, but classical programs which take problem size as a parameter and produce a quantum algorithm for each problem size, we have attempted to find scalable quantum algorithms with interesting complexity-theoretic implications. The bottleneck in genetic programming is the simulation of the quantum algorithm, which has potentially exponential slowdown; this has made it difficult to apply selection pressure for good performance on larger problem sizes. So far, the only scalable algorithms we have evolved are quantum simulations of simple classical randomized algorithms.

We have obtained more interesting results for particular instances of small problem size. These include instances of the Deutsch-Jozsa *early promise* problem, a database search problem, and the AND/OR problem. For the four-item database search problem our system evolved a solution that is essentially equivalent to the version of Grover's algorithm described in [19].

The AND/OR tree problem is the evaluation of a Boolean-valued property of a Boolean black-box (oracle) function; the quantum complexity of such problems has been studied, for example, in [20], [21]. For Boolean functions of binary strings of length $n$, the AND/OR property is a binary tree, having AND at the root and $n$ (including the root) alternating layers of Boolean OR and AND as nodes, with an $n$+1st layer of $2^n$ leaves consisting of the values of the black-box function ordered by their input string (viewed as a binary integer).

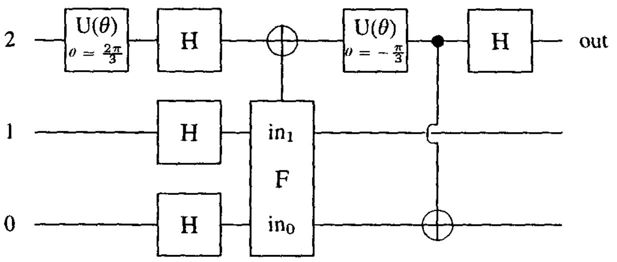

Figure 1: Simplified and hand-tuned version an evolved AND/OR algorithm. "F" = oracle function, final $\oplus$ = CNOT.

We applied genetic programming to the depth-two case, for which the problem is to evaluate:

$$(f(00) \vee f(01)) \wedge (f(10) \vee f(11)) .$$

Genetic programming (in a scheme without MEASURE gates) yielded an algorithm which uses only a single black-box function call and has error probability less than 0.41 for all black-box functions of two bits. We simplified and improved the algorithm by hand, producing that of Figure 1.

Both the hand-tuned algorithm and the evolved algorithm had better worst-case error probabilities than the simple classical probabilistic algorithm which consists of calling the black box function with a randomly chosen input, and returning the function value. The latter has worst-case error probability $\frac{1}{2}$ for six of the sixteen possible two-bit black box functions.

The hand-tuned algorithm has error 0 for the all-zero black-box function, $\frac{3}{8}$ for all other cases for which the correct answer is 0, and $\frac{1}{4}$ when the correct answer is 1. Unfortunately, this is not better than the best classical randomized algorithm, on the worst-case error criterion. The best classical Las Vegas algorithm (Las Vegas here means stochastic runtime, but guaranteed to give the correct answer) is quite simple, and Saks and Wigderson [22] showed it was the optimal algorithm for any depth tree. Santha [23] showed, for read-once Boolean functions, that no classical Monte Carlo algorithm with all error probabilities below $p$ can take fewer than $(1 - xp)Q$ queries, where $Q$ is the time taken by the optimal Las Vegas algorithm, and $x = 1, 2$ as the error is one- or two-sided. (This is just the "trivial" speedup obtained by flipping a biased coin to decide whether to do the optimal Las Vegas algorithm or output a random bit (two-sided) or a zero (one-sided), choosing the bias to achieve the desired error probability.) Thus a $q$-query quantum algorithm would have to have error probabilities $p < \frac{1}{x}(1 - \frac{q}{Q})$ to be better-than-classical. The best Las Vegas algorithm has worst-case expected queries 3 for the depth-two AND/OR tree, so a one-query quantum algorithm would need $p < 1/3$ two-sided, $p < 2/3$ one-sided to be better than classical, while a two-query algorithm would need $p < 1/6$ two-sided or $p < 1/3$ one-sided. Unfortunately, our one-query algorithm has two-sided error $.375 > 1/3$, and so does not beat the best classical

```
U-THETA [0] (5pi/11)
HADAMARD [1]
U2 [0] (-4pi 3pi/14 -pi/14 -12pi/13)
ORACLE [0 (in high) 1 (in low) 2 (out)]
HADAMARD [1]
MEASURE-1 [1]
HADAMARD [0]
U2 [0] (0 0 -pi/5 -2pi)
HADAMARD [2]
U-THETA [2] (11pi/9)
U-THETA [0] (3pi/4)
U-THETA [0] (4pi/15)
MEASURE-0 [0]
MEASURE-0 [2]
MEASURE-1 [2]
```

Figure 2: New evolved algorithm for AND/OR

| Function | Error Prob. | Function | Error Prob. |
|---|---|---|---|
| 0 0 0 0 | 0.0075.. | 1 0 0 0 | 0.2922.. |
| 0 0 0 1 | 0.2751.. | 1 0 0 1 | 0.2936.. |
| 0 0 1 0 | 0.2751.. | 1 0 1 0 | 0.2936.. |
| 0 0 1 1 | 0.2059.. | 1 0 1 1 | 0.2163.. |
| 0 1 0 0 | 0.2922.. | 1 1 0 0 | 0.2067.. |
| 0 1 0 1 | 0.2936.. | 1 1 0 1 | 0.2326.. |
| 0 1 1 0 | 0.2936.. | 1 1 1 0 | 0.2326.. |
| 0 1 1 1 | 0.2163.. | 1 1 1 1 | 0.0088.. |

Table 1: Error probabilities for the new evolved AND/OR algorithm

Monte Carlo algorithm on the worst-case error criterion. It is unclear whether its performance profile on all inputs could be achieved by classical means; the algorithms obtained by trivial speedup of classical Las Vegas cannot achieve or dominate it, since they have the same error probabilities for all inputs. The quantum algorithm essentially operates by applying a quantum routine to the lower order bit, with a value for the high-order input bit chosen randomly via "quantum dice"; the restriction of the algorithm to the low-order bit yields an algorithm for OR with one-sided error $1/4$, which does beat the Saks-Wigderson-Santha bound ($Q$= 3/2 for the depth-one case), but is dominated by a similarly derived algorithm described in the next section.

## 4 New Results

The new version of our genetic programming apparatus described in Section 2, including MEASUREMENT gates, evolved the algorithm listed in Figure 2. For the sixteen possible functions of two bits, this algorithm had error probabilities given in Table 1.

Changing some angles, combining some sequences of gates into single gates, and hand-optimizing the final rotation angle on qubit 2, yielded the algorithm of Figure 3. To

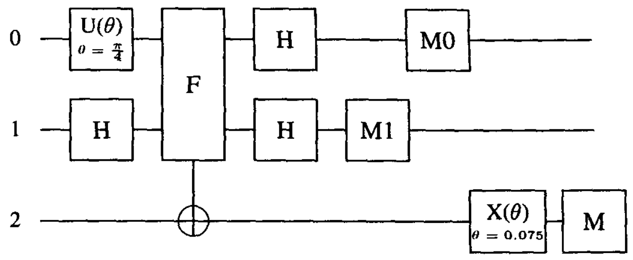

Figure 3: Hand-tuned version of new evolved AND/OR

| Orbit | $p_e$ (simplified) | $p_e$ (best evolved) |
|---|---|---|
| 0 0 0 0 | .00561 | .00830 |
| 0 0 0 1 | .28729 | .28873 |
| 0 0 1 1 | .21264 | .21547 |
| 0 1 0 1 | .28736 | .28858 |
| 1 1 0 1 | .21833 | .21957 |
| 1 1 1 1 | .00561 | .00830 |

Table 2: Error probabilities for new simplified AND/OR algorithm

avoid clutter in this and other algorithms, we define X $(\theta)$ = U2 $(\phi = 0, \theta, \psi = \pi/2, \alpha = \pi/2)$, with matrix:

$$\begin{bmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{bmatrix}$$

Note that the time-ordering of measurement gates on different qubits, indicated in our figures by the horizontal position of gates, is material to the algorithm. The $M$ gate indicates the sequence MEASURE-0, MEASURE-1 in immediate succession on the same qubit.

This simplified algorithm has error probabilities constant on orbits of the automorphism group, given in Table 2, which also reports error probabilities for another evolved algorithm which emerged while we were simplifying the first. The *automorphism group* of a Boolean property consists of those permutations of its input variables which leave the value of the Boolean property invariant, no matter what assignment is made to the variables (no matter what the black-box function is, in our context). Depth-two AND/OR has input variables $f_0 \equiv f(00), f_1 \equiv f(01), f_2, f_3$; its automorphism group is generated by the permutations of indices: $(0 \leftrightarrow 1), (2 \leftrightarrow 3), (0 \leftrightarrow 2, 1 \leftrightarrow 3)$. Application of a permutation from the automorphism group to a black-box function yields a function with the same AND/OR value; the six distinct orbits of this permutation group on the set of functions may be labeled $0000_1$, $0001_4$, $0011_2$, $0101_4$, $1101_4$, $1111_1$. The strings specify representative functions written $f_{00}f_{01}f_{10}f_{11} \equiv f_1f_2f_3f_4$; the subscripts indicate the number of functions in the orbit containing that representative. The first three orbits have AND/OR=0; the last three have AND/OR=1.

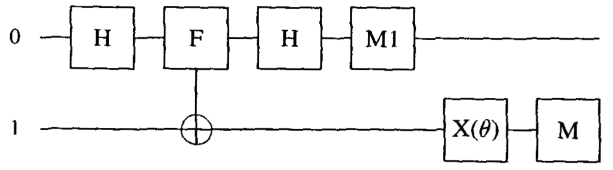

Figure 4: An algorithm for OR

## 5 Discussion

If we restrict the hand-tuned two-bit AND/OR algorithm to qubits 1 and 2 by fixing a value $x$ for qubit 0 and omitting the final two gates on it, we effectively get an algorithm for computing OR on the "marginal" black-box function of one bit $f^{(x)}$ defined by $f^{(x)}(y) = f(yx)$ (where $yx$ stands for concatenation, not multiplication). With qubits relabeled ($2 \rightarrow 1$, $1 \rightarrow 0$), the algorithm is given in Fig. 4.

Writing $c \equiv \cos(\theta)$, $s \equiv \sin(\theta)$, the error probabilities for this algorithm are

$$p_e^{odd} = 1/2 + \left(\frac{c-s}{2}\right)^2 , \; p_e^{even} = s^2 . \qquad (1)$$

With $\theta = 0$, this algorithm has error probabilities zero on the two even parity functions (the functions $f_0 f_1 = 00$ and $f_0 f_1 = 11$), and $1/4$ on the two odd parity functions. Since this is one-sided, it is below the classical Saks-Wigderson-Santha lower bound of $1/3$ for one-query routines. With $\theta = \sin^{-1}(\frac{1}{\sqrt{10}})$, it has error probability $1/10$ for all inputs, below the two-sided classical bound of $1/6$. (No exact one-query quantum algorithm for $OR$ is possible [24],[20].) The `MEASURE-1 [0]` gate may be replaced by a `CHADAMARD [0 1]` gate, which effects a Hadamard on qubit 1 if qubit 0 is in the state $|1\rangle$ and otherwise does nothing, yielding an algorithm with the same output probabilities but different characteristics when used as a building block in larger networks, since it does not gather information about (and thereby decohere) the two ways in which the algorithm of Fig. 4 may obtain the result 1.

A similar restriction of the algorithm to qubits 0 and 2 yields an analogous algorithm for AND on one-qubit functions. With $\theta = 0$ in both algorithms, the AND algorithm is equivalent (though not identical) to the AND algorithm obtainable from the OR algorithm via De Morgan's law. The overall algorithm may therefore be thought of as beginning with a "nested" use of the $\theta = 0$ algorithms for AND and OR, with the top-level AND algorithm, instead of calling a standard black-box oracle with two inputs superposed, calling the lower level OR routine on two functions ($f(0y)$ and $f(1y)$) in superposition. This interpretation must be used with care, however, since the "function" called by the top-level routine is not a standard black-box oracle or even a stochastic version of such an oracle, since we cannot measure the output of the lower-level algorithm before input to the top-level algorithm (which would make the lower-level algorithm behave like a stochastic black-box oracle) without destroying coherence on which the functioning of the top-level algorithm depends. Also, the nested algorithm is followed by an optimally-chosen $X(\theta)$ rotation before the final qubit. In simulations of other nesting schemes, we have seen cases where these additional interference effects in the top-level algorithm lead to a worse outcome (even with optimal rotation of the output qubit before the final measurement) than one would expect from a naive analysis which treats the lower-level routine as a stochastic black-box oracle. Here, these interference effects appear to improve the results. We will give a fuller treatment of the quantum mechanics of these algorithms in a future article.

While the depth-two AND/OR tree problem is only of theoretical interest, an efficient, scalable, quantum AND/OR algorithm could have much broader significance. AND/OR graphs have many applications in computer science; for example they have been applied to dynamic programming problems, decision tree optimization, symbolic integration, analysis of waveforms, and theorem-proving [25]. Considerable attention has been focused on AND/OR tree search algorithms such as heuristic AO*, and one would therefore expect better-than-classical quantum algorithms to have a major impact.

One particularly intriguing possibility derives from the observation that a Prolog program and query induce an AND/OR tree, the solution of which provides the answer to the Prolog query [26]. This suggests that an efficient, scalable, quantum AND/OR algorithm could possibly serve as the foundation for a better-than-classical Prolog interpreter, or "Quantum Logic Machine" (QLM). Because Prolog is a general-purpose programming language this further suggests a new approach to achieving quantum speedups for arbitrary algorithms, via expression of the algorithms in Prolog and execution on a QLM. We stress that this is only speculation at this time, as we currently have neither a scalable quantum AND/OR algorithm nor the full procedure for harnessing such an algorithm for better-than-classical Prolog query interpretation. But the idea of a QLM serves as a reminder that better-than-classical algorithms for AND/OR problems may someday have practical import.

## 6 Conclusions

Genetic programming appears to be useful in exploring the potential of quantum computation. We described how a quantum computer simulator and a specialized genetic programming system, both presented in some detail, can be used to discover new, better-than-classical quantum algorithms. In particular, we documented the evolution of a new better-than-classical quantum algorithm for the depth-two AND/OR tree problem. This new algorithm has several interesting properties and provides insights into the quantum speedups possible for Boolean oracle problems. We are currently working to leverage these insights and additional runs of our system to find a scalable better-than-classical AND/OR tree evaluation algorithm. Because of the many applications of AND/OR trees in computer science, we speculate that such a scalable algorithm would have many uses.

## Acknowledgments

Supported in part by the John D. and Catherine T. MacArthur Foundation's MacArthur Chair program at Hampshire College, by National Science Foundation grant #PHY-9722614, and by a grant from the Institute for Scientific Interchange (ISI), Turin. Some work reported here was performed at the Institute's 1998 Research Conference on Quantum Computation, supported by ISI and the ELSAG-Bailey corporation. We thank Harry Burhman, Richard Cleve, Michele Mosca, and Ronald de Wolf for valuable discussions and comments.

## Bibliography

[1] R. Jozsa, "Entanglement and quantum computation," in *Geometric Issues in the Foundations of Science*, S. Huggett, L. Mason, K. P. Tod, S. T. Tsou, and N. M. J. Woodhouse, Eds. Oxford University Press, 1997, http://xxx.lanl.gov/abs/quant-ph/9707034.

[2] Peter W. Shor, "Quantum computing," *Documenta Mathematica*, vol. Extra Volume ICM, pp. 467–486, 1998, http://east.camel.math.ca/EMIS/ journals/DMJDMV/xvo 1-icm/00/ Shor.MAN.ps.gz.

[3] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, S. Goldwasser, Ed. 1994, IEEE Computer Society Press.

[4] D. Beckman, A. N. Chari, S. Devabhaktuni, and J. Preskill, "Efficient networks for quantum factoring," Tech. Rep. CALT-68-2021, California Institute of Technology, 1996, http://xxx.lanl.gov/abs/quant-ph/9602016.

[5] L. K. Grover, "Quantum mechanics helps in searching for a needle in a haystack," *Physical Review Letters*, pp. 325–328, 1997.

[6] J. Preskill, "Quantum computing: Pro and con," Tech. Rep. CALT-68-2113, California Institute of Technology, 1997, http://xxx.lanl.gov/abs/quant-ph/9705032.

[7] A. Steane, "Quantum computing," *Reports on Progress in Physics*, vol. 61, pp. 117–173, 1998, http://xxx.lanl.gov/abs/quant-ph/9708022.

[8] S. L. Braunstein, "Quantum computation: a tutorial," Available only electronically, on-line at URL http ://chemphys.weizmann.ac.il/ ~schmuel/comp/comp.html, 1995.

[9] C. P. Williams and S. H. Clearwater, *Explorations in Quantum Computing*, Springer-Verlag, TELOS, 1997.

[10] G. J. Milburn, *Schrödinger's Machines: The Quantum Technology Reshaping Everyday Life*, W. H. Freeman & Co., 1997.

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

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

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

[14] Peter J. Angeline and K. E. Kinnear, Jr., Eds., *Advances in Genetic Programming 2*, MIT Press, 1996.

[15] Lee Spector, W. B. Langdon, Una-May OReilly, and Peter J. Angeline, Eds., *Advances in Genetic Programming 3*, MIT Press, 1999.

[16] L. Spector, H. Barnum, and H. J. Bernstein, "Genetic programming for quantum computers," in *Genetic Programming 1998: Proceedings of the Third Annual Conference*, J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. L. Riolo, Eds. 1998, pp. 365–374, Morgan Kaufmann.

[17] L. Spector, H. Barnum, H. J. Bernstein, and N. Swamy, "Quantum computing applications of genetic programming," in *Advances in Genetic Programming 3*, Lee Spector, W. B. Langdon, Una-May OReilly, and Peter J. Angeline, Eds., pp. 135–160. MIT Press, 1999.

[18] A. Ben-Tal, "Characterization of pareto and lexicographic optimal solutions," in *Multiple Criteria Decision Making Theory and Application*, Fandel and Gal, Eds., pp. 1–11. Springer-Verlag, 1979.

[19] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca, "Quantum algorithms revisited," *Proceedings of the Royal Society of London A*, vol. 454, pp. 339–354, 1998.

[20] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf, "Quantum lower bounds by polynomials," *FOCS '98*, pp. 352–361, 1998.

[21] Harry Buhrman, Richard Cleve, and Avi Widgerson, "Quantum vs. classical communication and computation," *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 63–68, 1998.

[22] M. Saks and A. Wigderson, "Probabilistic Boolean decision trees and the complexity of evaluating game trees," *Proceedings of the 27th IEEE Symposium on the Foundations of Computer Science (FOCS)*, pp. 29–38, 1986.

[23] M. Santha, "On the Monte-Carlo Boolean decision tree complexity of read-once formulae," *Proceedings of the 6th IEEE Structure in Complexity Theory*, pp. 180–187, 1991.

[24] R. Jozsa, "Characterizing classes of functions computable by quantum parallelism," in *Proceedings of the Royal Society of London A 435*, 1991, pp. 563–574.

[25] Nils J. Nilsson, *Principles of Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., 1980.

[26] Yoav Shoham, *Artificial Intelligence Techniques in Prolog*, Morgan Kaufmann Publishers, Inc., 1994.