Quantum Computation vs. Evolutionary Computation
Could Evolutionary Computing Become Obsolete?

Hugo de GARIS,
Ravichandra SRIRAM
Brain Builder Group,
Department of Computer Science,
Utah State University, Logan,
UT-84322-4205 USA
degaris@cs.usu.edu
ravichandras@cc.usu.edu

Zijun ZHANG
Department of Physics
Utah State University, Logan,
UT-84322-4205, USA
SLL34@cc.usu.edu

Abstract- This paper raises the rather provocative question as to whether the rise of quantum computing (QC) may, in time, make the field of evolutionary computing (EC) obsolete. This is not an idle speculation. Quantum computing is capable of processing huge numbers of quantum states simultaneously, in parallel, (“quantum parallelism”), whereas evolutionary computing can only process one chromosome per processor. In theory, QC ought to be able to process all possible points in a $2^N$ search space (of N bit chromosomes). If N is large, $2^N$ is gigantic, making a systematic search with a classical computer impossible. An evolutionary algorithm is effectively a guided search algorithm that samples the search space. Since QC can examine all $2^N$ points simultaneously, it looks as though EC is doomed to be replaced by QC, as soon as QC is successfully implemented in the future. This paper presents examples of two simple QC algorithms applied to the generation (c.f. evolution) of neural network circuits, which our group normally evolves using EC algorithms. The paper also reflects on whether EC might still survive under certain circumstances.

1. Introduction

The prime research interest of the first author is “artificial brains”. For the past decade or more, he has been attempting to evolve large numbers of neural network circuit modules, each with its own function, and then interconnecting them according to the designs of human “BAs” (Brain Architects) to build “artificial brains”. Until recently, this work had to be done in specialized “evolvable hardware” machines such as the “CBM” [3], that evolved modules in a few seconds each, compared to several hours on a PC.

Very recent software-based evolutionary algorithms, e.g. “LEM” [2], which accelerate the speed of neural net evolution by a factor of hundreds, allow graduate students to evolve hundreds of neural net modules within a semester and to build their own artificial brains [1]. This exciting development is not the only recent innovation, however.

The first author is not only a professor of computer science, but also an adjunct professor of theoretical physics in the physics department of his university, supervising a physics PhD student in the topic of quantum computing and has taught a computer science PhD class in quantum computing. Hence the first author and his research group have the necessary knowledge to explore the possibility of devising quantum computing algorithms to generate neural network circuit modules for building artificial brains. Since such brains might potentially be constructed with quantum neural networks, it seems reasonable to label such brains – “quantum brains”.

The impetus for the work presented in this paper originated in the following way. The first author was reading the references of a quantum computing text and happened to notice one concerned with “quantum arithmetic” (i.e. how to use quantum computing algorithms to perform addition, subtraction, multiplication (repeated addition), etc [9]). Since the classical computing algorithms used to evolve his neural networks involved only arithmetic, it suddenly occurred to him that it might be possible to “evolve” neural networks quantum mechanically.

This paper presents two such (simplified) quantum neural network generation (c.f. evolution) algorithms. They serve as a case study supporting the idea that QC may make EC obsolete. This is not an idea to be readily dismissed. It may very well happen.

The remaining contents of this paper are as follows. Section 2 gives a brief overview of the classical computing approach taken by our group to the evolution of neural net circuit modules. These techniques serve as the basis for two simple quantum versions presented in sections 3 and 4. Section 5 discusses ideas for future work, particularly the need for “quantum compilers”. Section 6 concludes and speculates on whether EC can survive at all in the longer-term future.
2. Classical Evolution of Neural Networks

This section gives a brief overview of classical neural networks and how our group has evolved them in the past. The basic ideas of this classical approach to neural network evolution are carried over to the quantum approach in the QNN algorithms of sections 3 and 4.

Our group’s classical approach employs fully connected neural networks of $N$ neurons, and hence $N^2$ connections, each of which is "weighted" by an $M$-bit binary fraction. These weights are evolved using a Genetic Algorithm (GA), to generate desired signaling behaviors for given external inputs. The signal values are real numbers that range between +1.0 and -1.0. An incoming signal vector (to each neuron) is scalar-producted with its weight vector, and an external signal value is added to this product. The sum is applied to a sigmoid function whose output is the output signal of the neuron that is transmitted to all other neurons. The “fitness” value is defined to be the measure of how closely the neural net output follows a given target. It is these fitness values that drive the evolutionary process.

3. A Quantum 1-Bit Signaling, 1-Bit Weight Neural Net Model, “QNN-1”

It is assumed in this paper that readers are already familiar with the basic principles of quantum computing. If not, then interested readers may consult the references [4-12]. This section describes a quantum-computing-based algorithm called QNN-1 that generates neural networks, which will serve as a model (or proof of concept) for future work. Our group aims to develop such quantum algorithms further, and to combine quantum generated neural circuit modules into quantum artificial brains. By a “quantum artificial brain” is meant that its components can be generated using future quantum based algorithms such as QNN-1, and that the resulting brain can be run using quantum mechanical principles. The construction of a quantum artificial brain would be somewhat analogous to the construction of a classical artificial brain except that the former approach would use quantum computing and the latter would use classical computing.

Consider Fig. 1,

a) $W_{11}, W_{22}, W_{12}, W_{21}$ are the (binary) weights of the specific links in the directions indicated by the arrows. $W_{11}, W_{22}, W_{12}, W_{21} = \{0,1\}$.

b) $NS_1$ and $NS_2$ are the output signals of the neurons. $NS_1, NS_2 = \{0,1\}$. At the 0th clock tick, $NS_1$ and $NS_2$ are 0. For all further clock ticks, the outputs of the neurons are $NS_1$ and $NS_2$ (for neuron 1 and 2 respectively).

c) $EXS$ is the external signal given to each neuron. $EXS = \{1\}$

d) The desired (or target) behavior of this extremely simple neural net circuit is to output the minimum number of zeros (i.e. the maximum number of 1’s) for all clock ticks.

3.1 Description of the Qnet Circuit:

Here we describe the Qnet Circuit (see Fig. 2), operator by operator, for one clock tick.

**Fig. 1 A Simple 2 Neuron Network**

**Step 1) Initialization of the States:**

In Fig. 2 the left most boxes represent the qubits of the Qnet Circuit.

i) The first four boxes $W_{11}, W_{22}, W_{21}, W_{12}$ represent the weights of the neural net that are prepared in a superposed state using Hadamard gates. These superimposed weights represent the whole search space of all possible chromosomes in genetic algorithm terms.

ii) The fifth and sixth boxes $NS_1, NS_2$ represent the signals output by the two neurons from the previous clock tick. Initially $NS_1$ and $NS_2$ are $|0\rangle$.

iii) The seventh box $EXS$ represents the external signal given to each neuron initially to start the signaling of the neural net.

iv) The remaining boxes represent the qubits in the pure state as indicated by each box, i.e. either $|0\rangle$ or $|1\rangle$. 
Step 2) Signaling through the Qnet Circuit.

**Operators 1 – 4:** The first step in implementing the neural net is to calculate the scalar (or dot) product of the weights and the signals using the equations

\[(W_{11}*NS_{1} + W_{21}*NS_{2}) \quad \text{--------(1)}\]

\[(W_{22}*NS_{2} + W_{12}*NS_{1}) \quad \text{---------(2)}\]

We applied four Controlled Controlled Not Gates (CCNot a.k.a. Toffoli), and fixed the target bit at |0>. The circuit will then behave like an AND gate (i.e., (a,b,0) => (a,b,a&b)). We used an AND gate to implement the scalar product because we have binary weights and signals (which can be either 0 or 1).

**Operators 5 – 8:** After calculating the scalar product, we add the external signal EXS to equation (1) above to get NS1 for the next clock tick. These operators when combined, perform the addition of the scalar product and the external signal. They take the external signal as the carry in and output the sum of the three terms and the carry. This quantum addition is based on the work of Vedral et al [m].

**Operators 9 – 12:** These operators do for NS2 and EXS what operators 5-8 do for NS1 and EXS.

**Step 3) Threshold Calculation**

After computing the output signal for each neuron, we decide next whether the neuron will fire or not by comparing the output signals we calculated from equations (1) & (2) (+ EXS) with some threshold value. By threshold value we mean that if the output signal value (two bits) of the neuron is greater than or equal to the threshold value, the neuron outputs a 1, otherwise it outputs a 0, which will be input to the circuit at the next clock tick. We take the threshold value to be 1 because then our input signal has only one bit (after obtaining the two signals we ORed them and accordingly we passed a 1 or a 0).

**Operators 13 – 18:** These operators when combined, perform the ORing of the two signal bits. We built an OR gate with the help of NAND gates and when we analyzed the Toffoli gate, we found that if we keep the previous target bit at 1 we get results similar to a NAND gate. To build an OR gate we used de Morgan’s law, namely \((A’*B’)’ = A + B\), i.e. 2 NOT gates and a NAND gate.
Step 4) Fitness Calculation

Here the fitness is defined to be the minimum number of 0’s output, for all the clock ticks. After comparing the signal with the threshold we need to store the output signals (i.e. the NS1 output signals of the neural net) to perform the fitness calculation afterwards, as well as pass them as neuron signals for the next clock tick. As the output signal of the network, only one output neuron signal is chosen (for all clock ticks). This raises the issue of copying signal values, which is not allowed according to the “No-Cloning Theorem”. (Since our signals are in superimposed states, there is no way to store them). We therefore place a composite adder in Fig. 2 which gives not only the sum of the signal outputs from all previous clock ticks, but also outputs the same signal that was input to it. This signal is the neuron output signal for the next clock tick. The “composite adder” consists of 6 half adders, as shown in Fig. 3.

Fig. 3 shows the internal circuit of the composite adder of Fig. 2. In this circuit the first block of six half adders represents the composite adder at clock T and the second block of six half adders represents the composite adder at clock T+1. Fig.3 shows how incoming information is both passed through the circuit and is added to the fitness sum. Six half adders are taken because we use 64 clock ticks. When we add 64 bits serially we get a 6 bit binary number ($2^6 = 64$). In Fig. 3 the first half adder on the left side will be given the output signal NS1 of a neuron as an input from clock T-1.

Signals S0 to S5 are the sum bits from clock T. The input NS1 is passed through a NOT gate so that if we get a 1 as output we give a 0 as input. We thus add the number of zero’s in the output signals, to find the fittest chromosome (with lowest fitness value). All the other inputs are qubits prepared in the pure state |0>. The output of this composite adder at clock T is given as input to the composite adder at clock T+1. We obtain some unused bits that we label GB (garbage bits). We repeat this process for 64 clock ticks, thus obtaining the sum of the output signals for each chromosome.

We can then apply “Durr’s Algorithm” [12] to find the index position of the minimum value in a given set. This index maps to the index of the weights. We can therefore find the weights of the fittest chromosome and use the corresponding neural network to perform the desired function.

Step 5) Recycling of the Qubits:

The operators 19 - 22 in Fig. 2 and 23 – 34 in Figure 4 are used to reverse the operations that were performed earlier, by running them in reverse order. This saves circuitry by allowing us to use the same wires. For a very clear explanation of quantum arithmetic circuits, see [9].
Fig. 4 A Circuit for Recycling the Qubits

Figure 5 Weight-Signal Multiplication Circuits

Figure 6 Circuit for Summing the Weighted Signals
4. A Quantum 1-Bit Signaling, 2-Bit Weight Neural Net Model, “QNN-2”.

This section extends the work done in section 2, to a 1-bit signaling, 2-bits weight model, called “QNN-2”.

The signal values (NS1 and NS2), the external signal (Ext or EXS) and the desired target behavior (i.e. the minimum number of zeros) are the same as in the basic neural net model described above (“QNN-1”). The complexity of the model is increased by giving the weights 2-bit values, and using a more complex threshold circuit. The weights are labeled W11, W22, W12, W21 = {0,1,2,3} and the threshold value is greater or equal to 8.

The neural net calculations are as shown in the circuits.

Figure 5 shows the Qnet circuits for the calculation of W11*NS1 and W21*NS2. Similarly we can calculate W22*NS2 and W12*NS1 where W11, W22, W21 and W21 represent the weights of the neural net that are prepared in a superposed state using Hadamard gates. The superimposed weights represent the whole search space of all possible chromosomes in genetic algorithm terms.

Figure 6 shows the calculation of W11*NS1 + W21*NS2 from W11*NS1 and W21*NS2, given as inputs. The composite gate (which has two + signs in the middle is a two bit adder, which takes two 2-bit numbers and one 0 bit to give the sum of the two 2 bit numbers. Its construction can be seen in [9].

Hence from the calculations of Figure (b) 6 we get

W11*NS1 + W21*NS2 --- (1)

and similarly we can calculate

W22*NS2 + W12*NS1 --- (2).

After calculating the scalar product, we add the external signal Ext to equation (1) shown above to obtain the intermediate value for calculating NS1 for the next clock tick. Figure 7 gives W11*NS1 + W21*NS2 + Ext. Similarly we can calculate W22*NS2 + W12*NS1 + Ext for the second neuron. The logic used in Figure 7 is to first add the Ext to the least significant bit and if any carry results to add it to the next significant bit, and so on, until the carry propagates to the most significant bit to give the output.

Figure 8 shows the threshold calculation. After computing the output signal (e.g., at neuron 1, W11*NS1 + W21*NS2 + Ext) we decide whether the neuron will fire or not by comparing the output signals we have just calculated at each neuron with a threshold value (>= 8, in this case). By a threshold value we mean that if the output signal value (four bits) of the neuron is greater than or equal to the threshold value (8, in this case), the neuron fires and outputs a 1 (which clears the accumulated sum), otherwise a 0 is output, and the current sum is accumulated at that neuron. It will be combined with the output signal of the neuron at the next clock tick, and if that accumulated sum is greater than or equal to 8, the neuron fires and clears the accumulated sum. For each neuron, one Figure 8 circuit is needed for each calculation and for each clock tick. The accumulated sum at each neuron is carried to the next clock tick based on the truth table shown in Figure 9.

Let ABCD represent the 4-bit output of the neural net (i.e., W11*NS1+ W21*NS2 + Ext value). If that value is greater than or equal to 8 then NS1 will become 1 as shown in the table. When the output of the neural value is greater than or equal to 8, at the next clock tick, the accumulated sum value (EFGH) will be zero, because the neuron fires at this clock tick. If the neuron outputs a zero without firing, then the accumulated sum is carried to the next clock tick until it become greater or equal to 8 (which shows the identical values of ABCD and EFGH if ABCD is less than or equal to 7). The fitness calculation is performed in the same way as in QNN-1, and the same type of composite adder can be applied here.
5. Future Work: “Quantum Compilers”

The above two quantum circuit models for single quantum bit signaling and single and double quantum bit weights are obviously extremely primitive quantum neural network models. It is obvious that the next immediate steps will be to devise more elaborate quantum neural net models, with multi-qubit signaling and weights, and more elaborate “fitness definitions”. (Note, by “fitness definition”, we are no longer thinking in terms of the usual use of the term within the context of evolutionary computation, because the whole point of this paper is to negate the need for an evolutionary (i.e. sampling) based computational paradigm. By “fitness definition”, we mean “performance measure”, “quality measure”, etc.)

But there is a problem when one attempts to generate quantum neural net circuits that implement more complex models. This should be clear from the above two circuit models, namely the tedium of dealing with individual quantum bits. An analogy can be made between the difficulty of dealing with bit level programming as existed in the 1940s (before the creation of assembler and compiler translation programs) and today’s individual quantum bit (qubit) level quantum computing. The reasoning used to justify the creation of higher-level classical computing languages applies to quantum computing as well, namely it is much easier to write statements in a high level language and have a translation program take care of the tedious generation of qubit level quantum circuitry than it is to have to think in terms of the qubit level circuits themselves.

Therefore our research group is thinking seriously about creating a quantum (neural network generating) compiler. This will involve the creation of a high level language appropriate for the generation of (quantum) neural networks. We will need to decide the functionality of the language. How general should such a language be? Should it be limited to the generation of single quantum (neural network) circuits or should it be...
able to handle multiple (i.e. interconnected) circuits, leading towards the creation of artificial “quantum brains”, consisting of large numbers of interconnected quantum neural net circuits. Once such decisions are made, work can begin on writing the compiler whose output will be the quantum circuits (quantum networks), with their many quantum bit lines and quantum operators. Such quantum compilers will remove the tedium from human “quantum programmers” of having to generate these quantum circuits by hand.

Once the quantum compilers generate the quantum circuits, it would be useful to have “quantum circuit simulators” to simulate the behavior of the circuits. Until real quantum computers (i.e. actual hardware-implemented quantum computers) become a reality, we will be limited to simulating the behavior of quantum circuits. These quantum circuit simulators will necessarily be much slower, in fact, exponentially slower, than true-implemented quantum circuits. Nevertheless, their existence will be useful. We will be able to simulate complex quantum-neural-network-based artificial quantum brains.

6. Conclusions and Speculations

This paper claims that the rise of quantum computing (QC) will make evolutionary computation (EC) obsolete, because QC can calculate the fitness functions of all $2^N$ possible N qubit (quantum bit) strings simultaneously using the phenomenon of quantum superposition of states. Hence the search-space sampling approach as used by EC will no longer be needed. QC can measure the performance quality (i.e. the fitness) of EVERY point (of $2^N$, in the search space corresponding to N-bit strings) and simultaneously, using the superposition principle of quantum mechanics.

However, is this conclusion warranted? Could there be a case for a continual need for EC (sampled) thinking? This section speculates on such a question.

At any given moment in the future, the then state of the art quantum computer will be able to manipulate some finite number N of qubits, hence any problem that requires the use of search-spaces with more than $2^N$ search points in the space, will need a sampling approach to explore the space, and hence the EC approach will be justified. However, it is probable that such reasoning will only remain valid for a decade or two while the value of N remains fairly small. As QC technologies improve, the value of N will probably climb and climb, until possibly reaching Avogadro’s number (a trillion trillion) and beyond. If N can reach Avogadro’s number, then $2^N$ will be so astronomically large, it will be difficult to imagine any problem that will not succumb to its power.

It is fascinating to speculate on what kinds of problems might be solvable (or not be solvable) using such massive computing capacities (assuming of course that the appropriate quantum algorithms can be found to solve them, which is another non trivial issue, since discovering new quantum algorithms is not an easy matter).

References: