A Hybrid Quantum Evolutionary Algorithm

Ravichandra Sriram
Brain Builder Group
Computer Science Department
Utah State University
Logan, UT 84322
ravichandras@cc.usu.edu

Hugo de Garis
Head, Brain Builder Group
Computer Science Department
Utah State University
Logan, UT 84322
http://www.cs.usu.edu/~degaris
degaris@cs.usu.edu

Zijun ZHANG
Brain Builder Group
Physics Department
Utah State University
Logan, UT 84322
Sll34@cc.usu.edu

Abstract

This paper proposes a Hybrid Quantum Evolutionary Algorithm. 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 up to all possible points in a $2^N$ search space (of $N$ bit chromosomes). An evolutionary algorithm is effectively a guided search algorithm that samples the search space and in general is a slow process. Since QC can examine all or some of $2^N$ points simultaneously, this Quantum Parallelism can be cleverly used to speed the Evolutionary Algorithms. This paper presents a Hybrid Quantum Evolutionary Algorithm that uses the Quantum Parallelism and another well-known Quantum Algorithm to speed up the Evolutionary Algorithms.

1. Introduction

The prime research interest of the second 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.

The impetus for the work presented in this paper originated in the following way. The second author was reading the references of a quantum computing textbook 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.

However, it soon became clear that the fitness values of “chromosomes” in each generation (in GA terms) used in the evolution of N (qu)bit chromosomes could be calculated simultaneously, due to the superposition of states inherent in the formulation of quantum mechanics. This gives the potential for the speed up for the Evolutionary Computation Algorithm. The later sections explain how this Quantum Parallelism can be cleverly exploited to speed up the Evolutionary Algorithms.

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 explaining the Hybrid Quantum Evolutionary Algorithm. Section 3 gives the Quantum Hybrid Evolutionary Algorithm. Section 4 compares the Pure Evolutionary and the Hybrid Quantum Evolutionary Algorithm. Section 5 shows the circuits for implementing the Hybrid Quantum Evolutionary Algorithm. Section 6 gives the conclusions and Future work. Section 7 gives the References for this paper.

2. Classical Evolution of Neural Networks

This section gives a brief overview of the 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 Hybrid Quantum Evolutionary Algorithm.

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.

The GA can be briefly described in the following way:

Let the fully connected neural has \( N = 20 \) neurons. There will be \( N^2 = 400 \) connections and each connection is weighted by an \( M \)-bit (8-bit) binary fraction. The Length of each chromosome will be \( N^2 \times M \)-bit (400 * 8 = 3200).

**Step (1)** Consider a population (in GA terms) of chromosomes. Let population size be \( P = 1024 \). Initialize the first population of chromosomes randomly.

**Step (2)** Evaluate the fitness (in GA terms) for each chromosome present in the population.

**Step (3)** Find the best chromosome (according to the measure of fitness). It is finding the Maximum/Minimum (depends on the target function) fitness corresponding chromosome.

**Step (4)** Keep the elite chromosome and throw the rest of the chromosomes and fill all the 1023 chromosomes with the elite chromosome and now perform mutations (one elite chromosome is not disturbed) for the 1023 chromosomes (Generally in the initial generations, more number of mutations are induced per chromosome and in the later generations the number of mutations per chromosomes will become 1 mutation per chromosome. i.e., in the initial generations it is a strategy to explore the search space more and in the later generations it is a strategy to exploit a specific search space area that has the highest probability of finding the best chromosome).

Note: The Mutation strategy mentioned in the step (4) is one of the strategies followed by our group. Specifically this strategy is mentioned here, as it would be useful for the Hybrid Quantum Evolutionary Algorithm that will be discussed in the later sections.

**Step (5)** Repeat Steps (2), (3) and (4) until the fitness of the elite chromosome is satisfying.

Generally (from our past experiences) it takes around 20,000 generations \( G \) to evolve a target signal. (For a complicated target, it will take even more number of generations to evolve and the number of generations that would take to evolve a target will solely depend on the target complexity).

The Significant steps that will contribute to the complexity are Step (2), Step (3) and Step (4) and correspondingly their value is as follows (The complexity shown below is for the total evolution):

Let \( F = \) Number of steps/time taken to calculate the fitness of one chromosome (all multiplications or weights and signals etc., included).

Step (2) takes \( P * (F) * (G) \) where \( P \), \( F \) and \( G \) introduced above.

Step (3) takes \( P * (G) \) steps.

Step(4) takes \( \text{some constant} = k \) \( * (P) \) \( * (G) \). On average \( k = 10 \) approximately.

Therefore the total complexity will be \( (P)(G)(F) + (P)(G) + (k)(P)(G) \).

The above equation can be approximated to \( (k)(P)(G) \).

### 3. The Hybrid Quantum Evolutionary Algorithm

As the name suggests, this algorithm is a hybrid model using the “Quantum Parallelism” to speed up the Pure Evolutionary algorithms.

**Step (1)** is same as in the above pure Evolutionary Algorithm.

**Step (2)** Take the superposition of states for the population of chromosomes generated in the Step (1) and calculate the fitness of all the chromosomes simultaneously using Quantum gates. (The calculation is shown in the next sections in which calculations are done using the Quantum gates).

**Step (3)** Find the best chromosome (according to the measure of fitness). Applying Durr’s algorithm can do it and we can find the corresponding elite chromosome.

**Step (4)** and **Step (5)** are same as in the pure Evolutionary algorithm described above.

Therefore the total complexity will be \( (F) + (\text{Square root}(P)) \) \( (G) + (k)(P)(G) \).

The above equation can be approximated to \( (k)(P)(G) \).
4. Comparison of the Hybrid and the Pure Evolutionary Algorithm

When the values (All the specific values shown above for the population size etc., are taken from the algorithms that we implemented before) are substituted and \( (P)(G) = 2048 \times 10^4 \) steps.

When the complexity equations are compared, the first factor i.e., for step (2) for Quantum Evolutionary Algorithm has \( (P)(G) \) order of speedup and the Step (3) has nearly quadratic speedup due to the usage of Durr’s algorithm for the hybrid model. The step (4) is same for both the Pure Evolutionary and Hybrid models. It turns out from the specific values from the above two sections that the Hybrid model will have around 100 orders of speedup. This speedup makes the Hybrid model interesting.

5. A Quantum 1-Bit Signaling, 1-Bit Weight Hybrid Quantum Evolutionary 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 how we can implement some steps in the Hybrid Quantum Evolutionary Model, 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 hybrid 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 Hybrid Quantum Evolutionary Algorithm, 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) W11, W22, W12, W21 are the (binary) weights of the specific links in the directions indicated by the arrows. W11, W22, W12, W21 = \{0, 1\}.

b) NS1 and NS2 are the output signals of the neurons. NS1, NS2 = \{0, 1\}. At the 0th clock tick, NS1 and NS2 are 0. For all further clock ticks, the outputs of the neurons are NS1 and NS2 (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.

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](image)

**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 W11, W22, W21, W12 represent the weights of the neural net that are prepared in a superposed state using Hadamard gates. These superimposed weights represent one generation of chromosomes in genetic algorithm terms (As the considered circuit is tiny, these four weights represented the total search space and it’s a coincidence).

ii) The fifth and sixth boxes NS1, NS2 represent the signals output by the two neurons from the previous clock tick. Initially NS1 and NS2 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
\[(W11*NS1 + W21*NS2) \quad \text{--------- (1)}\]

\[(W22*NS2 + W12*NS1) \quad \text{--------- (2)}\]

We applied four Controlled Controlled Not Gates (CCNot a.k.a. Toffoli), and fixed the target bit at \(|0\rangle\). The circuit will then behave like an AND gate (i.e., \((a,b,0) \Rightarrow (a,b,a\&b))\). 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].

**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.

**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 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.

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.

**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 \(|0\rangle\). 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. After finding the elite we can proceed to the next generation after inducing the mutations classically as explained in the Hybrid Quantum Evolutionary Algorithm in the above sections.

**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. 3 Internal Circuit of Composite Adder (Fitness Calculation)](image)

![Fig. 4 A Circuit for Recycling the Qubits](image)

6. **Conclusions and Future Work**

The above quantum circuit models for single quantum bit signaling and single 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”.

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 circuit models, namely the tedium of dealing with individual quantum bits. Therefore our research group is thinking seriously about creating a quantum (neural network generating) compiler. There are some Quantum Compilers exist today but a specific Quantum Compiler that would be useful in the development of Quantum Circuits to be used for the building of Artificial Quantum Brains is absolutely needed. 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 that would be useful for building Artificial Quantum Brains. 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” in the Artificial Quantum Brains development 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.

This paper proposed a Hybrid Quantum Evolutionary Algorithm and showed a method of how to use QC to speed up the Pure Evolutionary Algorithms that are otherwise very slower. The orders of speed up obtained were really encouraging. Presently, our group from this inspiration is working on an algorithm that can be completely Quantum and which could have more speedup than the present Hybrid model. We are in search of a Quantum operator that can directly perform mutations to the chromosomes (leaving the elite one) that are in the superposition of states. Once our group invents that operator, then the number of steps taken for inducing mutations quantum mechanically would be very less when compared to pure evolutionary algorithms and all the steps in the (hybrid) algorithm would be performed Quantum mechanically to go from one population to the next population until the fitness value of the elite chromosome is satisfying or until a predetermined number of generations are completed.

References


