Evolving Quantum Circuits using Genetic Algorithm

Martin Lukac Marek Perkowski

Department of Electrical and Computer Engineering
Portland State University
Portland, OR, 97201
USA
lukacm@ece.pdx.edu
mperkows@ece.pdx.edu

***Abstract**: In this paper we focus on a general approach of using genetic algorithm (GA) to evolve Quantum circuits (QC). We propose a generic GA to evolve arbitrary quantum circuit specified by a (target) unitary matrix as well as a specific encoding that reduces the time of calculating the resultant unitary matrices of chromosomes. We demonstrate that, in contrast to previous approaches, our encoding allows synthesis of small quantum circuits of arbitrary type, using standard genetic operators.*

1. Introduction

While quantum mechanics and quantum computing are quite established research areas, automated quantum circuit synthesis is still only at the beginning of its exploration [2,4,6,7,8]. In quantum computation we use quantum bits (q-bits) instead of classical binary bits to represent information. This gives the advantage of being able to perform massively parallel computations in one time step. The design of quantum circuits of practical size is still technologically impossible, but the progress is fast and there are no arguments based on physics against the possibility of building powerful quantum computers in future. Therefore quantum computing area of research is recently flourishing. Finding an effective and efficient method of designing QC can be used for two applications: (1) modeling quantum computers in FPGA-based reconfigurable hardware for speeding-up computations that are very inefficient on standard computers [9], and (2) designing new optimized gates and circuits for theoretical investigations and for use in future quantum computers.

The major difference between quantum logic and binary logic is the concept of the information itself. While the classical (binary or multi-valued) representations of information are precise and deterministic, in Quantum Computing the concept of bit is replaced by the q-bit. Unlike classical bits that are realized as electrical voltages or currents present on a wire, quantum logic operations manipulate q-bits [7]. Qubits are microscopic entities such as a photon or atomic spin. Boolean quantities of 0 and 1 are represented by a pair of distinguishable different states of a qubit. These states can be a photon's horizontal or vertical polarization denoted by $|\leftrightarrow\rangle$ or $|\updownarrow\rangle$, or an elementary partice's spin denoted by $|\uparrow\rangle$ or $|\downarrow\rangle$ for spin up and spin down, respectively. After encoding these distinguishable quantities into Boolean constants, a common notation for qubit states is $|0\rangle$ and $|1\rangle$.

Qubits exist in a linear superposition of states, and are characterized by a wavefunction ψ. As an example, it is possible to have light polarizations other than purely horizontal or vertical, such as slant 45° corresponding to the linear superposition of $\psi=½[\sqrt{2}|0\rangle+\sqrt{2}|1\rangle]$. In general, the notation for this superposition is $\alpha|0\rangle+\beta|1\rangle$. These intermediate states cannot be distinguished, rather a measurement will yield that the qubit is in one of the basis states, $|0\rangle$ or $|1\rangle$. The probability that a measurement of a qubit yields state $|0\rangle$ is $|\alpha|^2$, and the probability is $|\beta|^2$ for state $|1\rangle$. The absolute values are required since, in general, α and β are complex quantities.

Pairs of qubits are capable of representing four distinct Boolean states, $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$, as well as all possible superpositions of the states. This property is

Proceedings of the 2002 NASA/DOD Conference on Evolvable Hardware (EH'02)
0-7695-1718-8/02 $17.00 © 2002 IEEE

known as "entanglement", and may be mathematically described using the Kronecker product (tensor product) operation ⊗ [7]. As an example, consider two qubits with $\psi_1=\alpha_1|0>+\beta_1|1>$ and $\psi_2=\alpha_2|0>+\beta_2|1>$. When the two qubits are considered to represent a state, that state ψ_{12} is the superposition of all possible combinations of the original qubit, where

$$\psi_{12}= \psi_1\otimes\psi_2 = \alpha_1\alpha_2|00> + \alpha_1\beta_2|01> + \alpha_2\beta_1|10> + \beta_1\beta_2|11>. \quad (1)$$

Superposition property allows qubit states to grow much faster in dimension than classical bits. In a classical system, *n* bits represents 2^n distinct states, whereas n qubits corresponds to a **superposition** of 2^n states.

In terms of logic operations, anything that changes a vector of qubit states can be considered as an operator. These phenomena can be modeled using the analogy of a "quantum circuit". In a quantum circuit wires do not carry Boolean constants, but correspond to pairs of complex values, α and β. Quantum logic gates of this circuit map the complex values on their inputs to complex values on their outputs. Operation of quantum gates is described by matrix operations. Probabilistic calculations based on this representation are used in only very small quantum computers so far, but it was verified that information can be represented as a superposition of states of single q-bits, and that in one time step operations can be performed on several q-bits. Beside this useful effect of quantum computing, various other effects resulting from q-bit encoding emerge, such as q-bit entanglement. Moreover it was shown [7] that any QC has to be reversible. In this paper we focus only on the synthesis of arbitrary quantum circuits (and quantum gates in particular) of small size. We propose a generalized approach to the problem of QC synthesis by using a simple encoding and a generic GA without any problem-specific operators. Our results show that, in contrast to published work [4,6], any kind of genetic operators can be used by using the proposed encoding.

This paper is divided into seven sections. Section 2 gives a brief overview of genetic algorithm and quantum gates used in our experiments. Section 3 explains the new problem encoding that we devised, and section 4 the fitness function. Section 5 discusses our selection method and section 6 the experimental results. Finally section 7 concludes the paper.

1.GA for QC synthesis

This section presents a brief description of the GA as the generator of QC. GA is one of a widely used search heuristics; it is based on the principle of evolutionary computation. For each problem we define a population (a set of solutions) that are evolved under certain constraints. Here, each individual in the population will be a QC. The quality of each solution is evaluated by a fitness function. Here the fitness function is based on an entry-by-entry comparison between the entries of the individual's unitary matrix and the entries of the matrix of the target gate or circuit. Each individual in the population represents a chromosome. It encodes a particular QC. To evolve the initial population to a set of individuals with better properties, GA uses three types of genetic operators on chromosomes: mutation, crossover and reproduction. Each of these operators is applied to one or more individuals in order to increase their fitness values. In the GA presented here only three operators have been implemented, but we experimented with larger sets of operators.

Mutation represents a completely random operator that introduces noise in the current population by randomly modifying a part of a chromosome. This operator is very important while searching the problem space in order to avoid being trapped in local minima of the fitness function F. In a standard bit-wise encoding of the solution candidate as a chromosome, the mutation operator acts by inverting values of individual bits. Here units of the chromosome are the (elementary) quantum gates (QG) such as Hadamard Gate or Swap Gate. The mutation can change gates. With the encoding used here, possible results of this operator are the following: changing one QC to another one, adding a new QG, removing an existing QG, changing the placement of an existing QC in the QC, and changing one QG to another one.

Crossover operator is a tool that directly recombines existing QCs in order to explore a larger problem space. It cuts two chromosomes (two individuals) in one or more locations and swaps their parts.

Selection is based on the Stochastic Universal Sampling rule, [1]. Mutation and crossover are executed sequentially, based on two randomly generated numbers; mutation is applied with the probability exceeding 0.4 and crossover with the probability exceeding 0.7.

2.Encoding

While encoding a QC as a string of objects one encounters the following question: how to encode the number of wires and the position in the circuit of a gate in the least complex data structure possible? The encoding of the

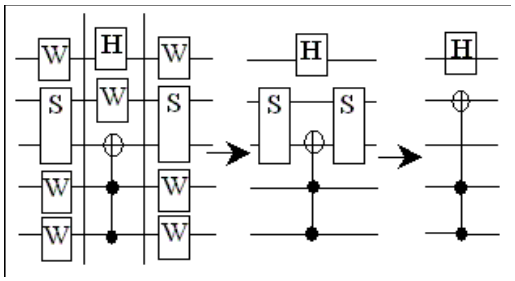


Figure 1: Transformation of a QC from the encoded chromosome (on the left) to a final quantum circuit notation representation of this

QC (on the right). Here S is a Swap gate, H is a Hadamard gate, W is a wire. In the middle there is one CCNOT (Toffoli) gate.

individual's chromosome is, in addition to the fitness function, one of the most important aspects of designing a well-converging GA. For that purpose we considered various encoding schemes [4,7], to finally come up with a simple and effective encoding for basic QC. This encoding is however powerful enough to allow describing any possible configuration of a QC. In contrast to the previous work, we do not use any additional specifications or parameters to specify the chromosome-defined QC. The order of consecutive quantum gates as well as a special parsing of the chromosome contains all information about the represented circuit. In this paper, there is no particular attention paid to explore the problem of changing parameters for certain gates (control bit) [4], but the focus is more on the global synthesis aspects of QC and the generality of the approach. While most of previous works use genetic programming, ours is only the second one that uses a GA and the first in which a standard GA is applied. An efficient encoding of each individual that can be then computed in parallel can consequently lead to the decrease of the computation time, thus making the GA more adapted to real-time quantum circuit design tasks. This encoding will have therefore applications in parallel computing and Evolvable Hardware accelerators for quantum circuits synthesis [9]. The conditions imposed on our encoding are the following: preservation of equal probabilities of presence of each type of gate, fast encoding and decoding of an individual gene in the chromosome and no parameters beside basic definitions (no control bits).

An example of our encoding is shown in Figure 1.

On the left side of Figure 1 it is shown how the circuit on the right of the same figure is encoded. As can be seen, there is no free space in the proposed encoding. Each place in the circuit is presented as a symbol of a unitary matrix of certain elementary quantum gate. A wire has a unitary identity matrix representation. While evaluating the fitness function, Kronecker products (tensor products) are executed on matrices of parallel gates (blocks, circuits), and standard matrix multiplications are performed on serial connections of gates. Each QC is parsed in parallel blocks, evaluated separately and

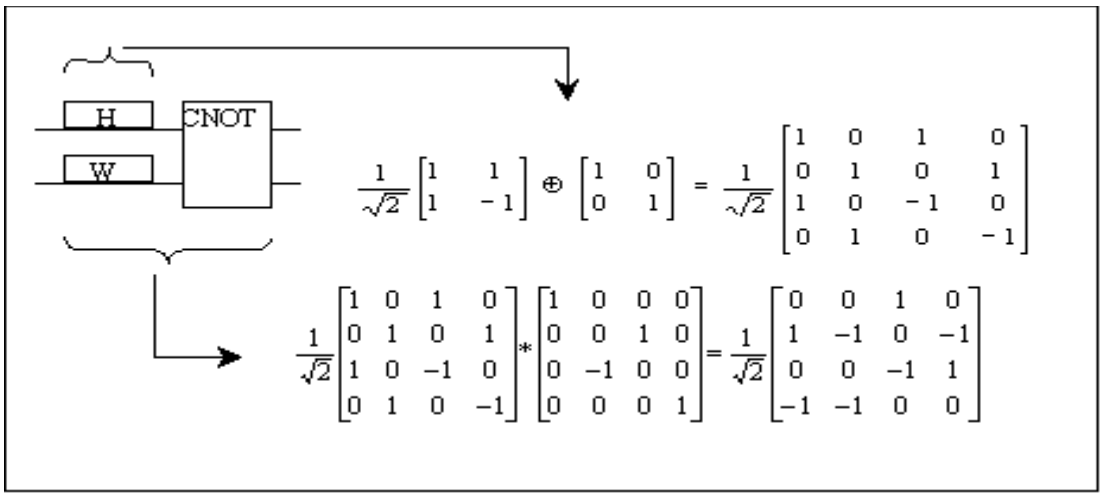


Figure 2: Examples of Kronecker product ⊕, and of Matrix product * on a sample of a circuit. H is Hadamard gate, W is wire and CNOT is Feynman.

finally multiplied together to give the final unitary matrix representation of the QC. This final matrix is next compared with the target matrix to evaluate their distance as a part of fitness function calculation. The example shown in Figure 1 illustrates a common inconvenience of encoding quantum circuits for genetic algorithms. A quantum gate CCNOT can be placed over three different arbitrary wires in a quantum circuit. However with the encoding used, there is no information indicating what gates are connected to what wires, beside the order of the gates. To solve this problem we insert two Swap gates (one before and one after) the CCNOT. This implies that outside of the Swap gates the CCNOT seems like being on wires 2,4 and 5, but the real CCNOT gate uses wires 3,4 and 5. In order of being able to encode a QC without any additional parameters, the circuit is split into parallel blocks where each block can be evolved separately.

This implies the fact, that each quantum circuit can be parsed into parallel blocks, where each block contains a series of ordered QC. Each parallel block is a small QC and can be used for mutation or crossover. No additional evaluations of blocks other than calculations of unitary matrices are necessary. For the above particular individual in the population, its chromosome representation (encoding) will be as in Figure 3.

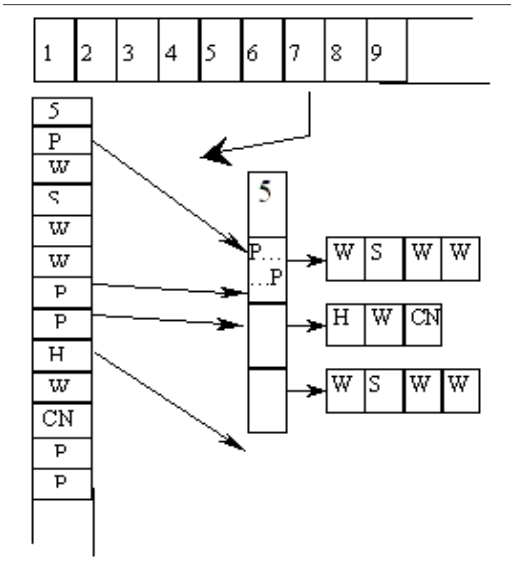


Figure 3 : Representation of an individual in the population (upper field), The chromozome encoding (on the left and bottom) and each parallel segement representation. W is wire, S swap gate, H Hadamard gate, CN Cnot (Feynman) gate. Letters P determine the border of each parallel segment. As QC does not have fan out or fan in, each parallel segment has the same number of input and outputs.

The mutation operator is not acting upon the gate definition itself, but can only move blocks of input/output vectors inside the chromosome. In the reported work we used only some of the possibilities of this operator, as presented above. Other possibilities will be investigated in the forthcoming work.

The genetic operators defined here will have the following properties:

- Mutation can change any gate to arbitrary other gate, with the same or different number of inputs/outputs.
- Mutation can induce a removal of a gate by changing it to a wire, or can simply remove an entire parallel block.
- Mutation changing a gate with n inputs to a gate with m inputs can add more gates or remove more gates in order to satisfy the number of inputs/outputs of the whole QC.
- Mutation plays here a major role. As the described below, the Crossover operator is quite constrained, the probability of mutation is very high [0.2, 0.8].
- Crossover can act upon a whole parallel block (from one chromosome) by interchanging it with another parallel block (from another chromosome) of the same number of inputs, i.e. with same number of wires. Crossover was used in this paper with probabilities [0.2, 0,8].

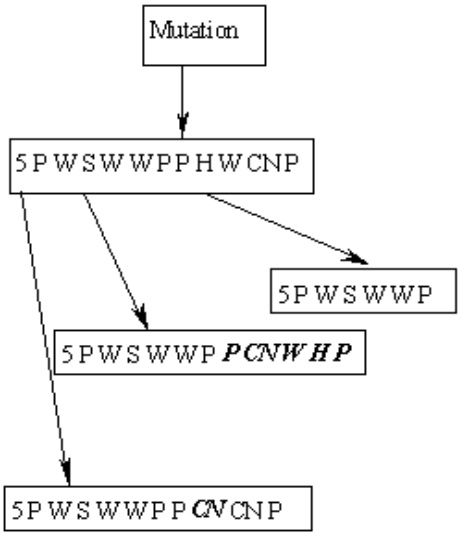


Figure 4: Example of results of a mutation on a chromosome. The complexity of the individual matrix of each circuit depends directly on the length of the chromosome and the number of wires.

3. Fitness function

Another important aspect of GA is the fitness function. Good design of the fitness function is crucial to the convergence of the whole quasi stochastic search process. It assigns fitness to each individual (chromosome) according to the evaluated error. Example: for a minimization of a function the individual whose chromosome has the smallest value error, will have the highest fitness. There is a magnitude of possible ways how to select the fitness function. Here we have the advantage that we know the goal (the "target matrix" or the matrix of the circuit to be synthesized), so we can easily determine the maximum fitness value. The result of this evaluation will be a "1" if all elements of the final matrix are the same as in a individual's chromosome, and less than 1 otherwise, proportional to the operator Λ, as described below.

The evaluation of an individual is done here similarly to [4,6] using an entry-by-entry comparison of the unitary matrices. However instead of looking for the global minimum we search for the global maximum by searching for the maximum value of $F = \frac{1}{1+error} - \Lambda$. The evaluation function for error is similar to [4,6]:

with U_{ij} is the ij[th] element of the requested unitary matrix for the searched circuit, and S_{ij} is the same element in the current matrix. We add another operator Λ to the fitness function, a direct factor influencing the fitness of the current circuit.

$$\Lambda_i = \begin{cases} \frac{\Phi_i}{n_i} & if\ \Phi_i < n_i \\ 0 & else \end{cases}$$

with Φ is the number of minimum parallel segments, and n is the total number of segments in the chromosome i. Note the minimum number of segments is a user-modifiable parameter to investigate trade-offs between the components of the fitness function *F*. Its function is to adjust fitness of an individual regarding its length. With respect to the evaluation process of any QC, an infinite number of circuits can be found with the same unitary matrix but with different structures; here we want to find the minimal circuit, that satisfies error equal zero. Value of Φ here was taken from the following value interval [1 - 5].

4. Selection Operator

With the selection operator, the GA chooses individuals for applying replication. Each individual selected by this operator will be either a parent for possible offsprings (using crossover) or will be just used in the next generation. Roulette Wheel rule is based on the fitness of each individual in the population, selecting the fittest individuals. The weakness

$$error = \sum_{i=1}^{2^n} \sum_{j=1}^{2^n} \left| U_{ij} - S_{ij} \right| \quad S, U \in U(2^n)$$

of this rule is that it contributes to a faster elimination of the non-fit individuals, and thus provides a fast convergence but with the weakness of sometimes not exploring the unknown part of the problem space. Stochastic universal sampling [1] is a method similar to the Roulette Wheel, but the selection of individuals is less biased by the fitness of the individuals. To select n individuals for replication, n pointers, each spaced by 1/n, are placed over the chromosome. The position of the first pointer is selected randomly. Each pointer will then point to a single individual, which will be next selected for replication. We compared both types of selection operations, resulting in stochastic universal sampling being used for most experiments.

5. Experimental results

The difficulties of applying GA for designing correct (and hopefully minimal) quantum circuits are the following. (1) A high time of evaluation of QC matrix, especially due to the calculation of Kronecker (tensor) product with sizes of matrices growing exponentially for larger circuits. This means that for the computation of large matrices one needs chromosome encoding that would allow for parallel computation. (2) If a high number of individuals are used for the total population, then the result can be find out in less generations [3] but with longer times of fitness evaluation. (3) Using a precise encoding for each specific configuration of a particular q-gate is obviously a big loss of time. To avoid extreme time consumption for calculations, the GA was limited to a relatively small population of individuals (50 – 100). Next we limited our explorations to circuits with the maximum of 4 wires, so that we can observe the time difference between the calculation of big (4 wires) and small (1 wire) unitary matrices. Finally to speed up the whole process we used OOP (object oriented programming) language so as to homogenize the programming and genetic operators.

We set up a set of tests, described below as the first step of setting up a basic library of benchmarks for automated QC design methods

In all examples discussed here, a one-point crossover approach is used. The mutation operator can erase, add, increase or decrease the chromosome length. To test the settings, all results have been averaged over 20 runs

Number of inputs	***Gates***
1	Wire, Hadamard, Pauli (X, Y,Z), Phase
2	Cnot, Swap, C-Z, C-phase
3	Ccnot(Toffoli), C-swap(Fredkin)

total, for each type of q-gate.
The GA was tested on two types of starting sets. Unrestricted starting set contains all gates

Table 1: Quantum Gates used in this experiment. (as building blocks of the chromosome and target gate) to be publicly available by researchers in this area. Our goal was to test evolutionary techniques for automated design of an arbitrary q-gate in the minimum time.

available, even if the search gate is among them. This was the major part of our experiments, because we wanted to discover similar circuits. Second approach is the restricted set, where only a random number of gates is selected, and the gate being searched is not included.

Table 2 shows the result for the first benchmark. For each GA run, one randomly selected quantum gate was used to create its unitary matrix, to be next used as a target matrix for the algorithm. The goal of the GA was to find at least the same gate, if not a similar and smaller but of the same functionality. The starting set of available gates to the GA was non-restricted (Table 1) and any number of gates was used. This test was set to test the convergence of our approach.

Table 2: Results of experiments. Due to the similarity of results we grouped the results by the number of inputs/outputs of the requested q-gate. PM and PC are probability of mutation and crossover.

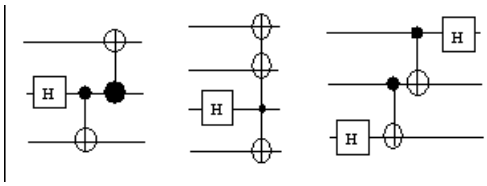
The results are quite encouraging. In every case the GA found the requested gate, however in no case the automatically created chromosome was better than the circuit for

Number of inputs per q-gate	Number of generations	pM	pC	Real time (average 20 runs)	pM<0.2 Number of generations	Real time (average 20 runs)	Population size
1 – input	<50	0.4	0.6	< 30 seconds	<100	< 1 minute	50
2 – inputs	<50	0.6	0.4	< 30 seconds	<100	< 1 minute	50
3 - inputs	50 - 200	0.6	0.6	<1 minutes	<200	<3 minutes	60

which the corresponding target unitary matrix was created. Summary of results is shown in Table 2.

All circuit evolved were exact copies or at least had same number of wires, of the searched circuit. For small gates the convergence is logically faster, because of the restricted recombination between different gates. A gate with more inputs than the number of wires in the circuit was not tested. The 3 input gates test shows the same result, however with exponential time. The results are measures of average values over 20 runs. Depending on the circuit we were looking for, the times are, as predicted, increasing very fast with the increase of the number of QC inputs. Two configurations were tested. First, high probabilities of mutation (0.4-0.7) and crossover (0.4-0.6) were applied. The results are surprising because of so high mutation probability. On the other hand the mutation here increase the recombination of gates, and allows searching more efficiently the problem space. The size of the GA population was set in range [50,100]. The very high probability of mutation allows a fast dangerous search, while compared to classical settings there is no proof of convergence. However the runs were stopped as soon as a good solution was found, even if only by a random search. A large random search with mutation used on a recombination problem seems to have a positive effect in a restricted search. The solution was found also when the mutation was of a small order, however the time of search raised as well.

Next step was to test composite circuits proposed by [4,6]. We selected three of them shown in Figure 4. The first two are both circuits to produce EPR as in [4], the last is the "send" circuit originally proposed by [10] and evolved in [6].



Number of inputs per q-gate	Number of generations	pM	pC	Real time (average 20 runs)	pM<0.2 Number of generations	Real time (average 20 runs)	Population size
3 - input	<150	0.4	0.6	< 1 minutes	<300	< 2 minute	50
4 - inputs	<350	0.6	0.4	< 2 minutes	<900	< 3 minute	50

Figure 4: 3 types of circuits searched with the GA.

The results are shown in Table 3. We were able to find all searched circuits, however in this part of experimentation the starting set of gates was open. Our GA found for all benchmarks at least similar, if not better, results compared to the published results of the studied cases. Even if the number of generations grows exponentially, the real time still remains reasonable.

The 3- and 4- input circuit search was made

Table 3: Results of benchmark tests for assembled circuits.

under similar conditions as the first part of experiments. Results from both tables shows that GA can be very successfully used to synthesize circuits. The time can be reduced by appropriate hardware and consequently used for still larger designs.

This allows us to claim that our algorithm is better then those previously proposed.

6. Conclusion

We have shown that the evolutionary computation can be used for automated QC development in real time using standard PC computers. Designed as shown, this algorithm can be also easily implemented on parallel computers or in classical binary FPGA-based evolvable hardware. An interesting research will be to implement evolutionary learning in a future truly quantum hardware which will lead to a new area of Evolutionary Quantum Hardware (EQH). Our program found one new circuit that was earlier came across by Williams [7] and three circuits located by Rubinstein [4]. In all cases that we studied the program was faster than the results previously published. In contrast to previous works that concentrated on some particular types of circuits such as teleportation [7] and entanglement [4] our approach is fully general. For instance the optimized version of the "send" circuit found by Williams was created. We will further experiment with the algorithm trying to find various realizations for gates and circuits from [8,9,10,12,13,14].

Our algorithm and its data structure can be applied without any modification to reversible circuits from "pseudo-classical" circuits [14]. Such circuits are used for instance in the famous Grover's Quantum Search Algorithm [11].

Reversible gates realizing Boolean operations can be realized not in quantum but in several other reversible technologies such as DNA, single-electron transistor, mechanical nano-switches, quantum dots or CMOS.

Although all benchmarks proved the convergence of our GA and results were better than previous ones, the goal of our approach is not only to benchmark a GA, but mainly to explore various evolutionary and other approaches [Alan,Andrey] to reversible and quantum circuit synthesis. Next step is to apply different Evolutionary algorithms such as Baldwinian or Lamarckian GA, genetic engineering or Evolutionary strategies. The encoding used here fits well also the design of reversible logic, where each parallel block in the chromosome is a small sub-circuit, with same number of wires as its neighbors. Obtained results also show the problem of scalability of any designing method. Here for 5 wires the time of search is small, and as previously said hardware implementation of a GA will speed the process up.

References

1. D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning *Addison Wesley*, 1989.

2. Y. Z. Ge, L. T. Watson, and E. G. Collins. Genetic algorithms for optimization on a quantum computer. In *Unconventional Models of Computation*, pp. 218-227.

3.Kuk-Hyun Han, Kui-Hong Park, Ci-Ho Lee, and Jong-Hwan Kim, "Parallel quantum-inspired genetic algorithm for combinatorial optimization problems," *In Proceedings of the 2001 Congress on Evolutionary Computation,* volume 2, pp. 1422-1429, 2001.

4.B.I.P. Rubinstein, "Evolving quantum circuits using genetic programming", *Proceedings of the 2001 Congress on Evolutionary Computation (CEC2001),* pp. 144-151 (2001)

5.L. Spector, H. Barnum, H. J. Bernstein, and N. Swamy, "Finding a better-than-classical quantum AND/OR algorithm using genetic programming," In *Proceedings of the 1999 Congress on Evolutionary Computation,* volume 3, pp. 2239-2246, Washington D.C., 6.-9. July 1999. IEEE, Piscataway, NJ.

6.C.W. Williams, Gray G. Alexander, "Automated Design of Quantum Circuits", *QCQC '98,* Springer-Verlag, pp. 113-125 (1999)

7.Williams C. P., Clearwater S. H., "Explorations in Quantum Computing", *Springer-Verlag*, New York Inc. (1998)

8.T. Yabuki and H. Iba. "Genetic algorithms and quantum circuit design, evolving a simpler teleportation circuit," In *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, pp. 421-425, 2000.

9.G. Negovetic, M. Perkowski, M. Lukac, A. Buller, "Evolving quantum circuits and an FPGA-based Quantum Computing Emulator," submitted, 2002.

10.Brassard G., Braunstein S. L., Cleve R., "Teleportation as Quantum Computation", *in Proceedings of the Fourth Workshop on Physics and Computation*, New England Complex System Institute.

11.L.K. Grover, "A Framework for Fast Quantum Mechanical Algorithms," *ACM Symposium on Theory of Computing* (STOC), 1998.

12.A. Barenco et al., "Elementary Gates For Quantum Computation", *Physical Review A* **5**2, 1995, pp. 3457-3467

13.M. Nielsen & I. Chuang, *Quantum Computation and Quantum Information*, Cambridge Univ. Press, September 2000.

14.T. Hogg et al., "Tools for Quantum Algorithms", http://arxiv.org/abs/quant-ph/9811073, 1998.