Genetic Algorithm Based Synthesis of Multi-Output Ternary Functions Using Quantum Cascade of Generalized Ternary Gates

Mozammel H.A. Khan,
Dept. of Computer Science and Engineering,
East West University, 43 Mohakhali, Dhaka 1212,
Bangladesh, mhakan@ewubd.edu

Marek Perkowski
Dept. of Electrical and Computer Engineering
Portland State University, Portland, OR 97207
mperkows@ee.pdx.edu

Abstract -Ternary quantum circuits have recently been introduced to help reduce the size of multi-valued logic for multi-level quantum computing systems. However, synthesizing these quantum circuits is not easy. In this paper we describe a new genetic algorithm based synthesizer for ternary quantum circuits. Our results show some of the synthesized circuits use fewer gates than previously published methods.

I. INTRODUCTION

Quantum computing (QC) is a very promising and flourishing research area [9],[10],[16]. QC theoretically allows designers to build much more efficient computers than the existing classical ones. For example, some problems that can't be solved in polynomial time using classical computers can be solved in polynomial time using quantum computers [9] (proven already experimentally but for small data only). In part, this is because quantum circuits are inherently able to perform massive parallel computations [9],[10],[16]. While most of the results are for binary quantum computers, the multi-valued quantum logic synthesis is a very new research area. Unfortunately, previous synthesis methods produced circuits that were unnecessarily complex. One promising approach for reducing the circuit size is to use gates that are ternary counterparts of the classical binary Feynman gates and new 2-qudit ternary controlled gates (*qudit* is a multiple-valued counterpart of binary quantum bit or qubit).

The success in the true realization of some ternary permutation gates now allows us to physically build ternary quantum computer using these gates. However, synthesizing quantum circuits is not a trivial problem and most previous attempts have been disappointing. Although there are several papers about using GA for binary quantum computers [13],[14],[17],[18] and quantum inspired evolutionary algorithms [19], to the best of our knowledge, no attempts have been made to use GAs for designing ternary quantum circuits. This is the first paper to introduce a practical synthesis approach to synthesize directly with *generalized ternary gates* (GTG) gates and not only with Toffoli-like gates built on top of GTG gates [5],[6],[8]. This allows us to obtain significant reductions in terms of elementary gates that are directly realizable in ion trap technology.

The paper is organized as follows. In Section II we describe some previous work in multi-valued logic. Section III covers the fundamentals of multi-valued logic along with some key definitions. Section IV introduces some basic ternary quantum circuits. The general model for synthesizing multi-output ternary functions is given in Section V. Section VI provides details on our GA. Section VII discusses a new feature of our approach – synthesis of incompletely specified functions and Section VIII the experimental results. Section IX concludes the paper and Section X presents future work.

II. PREVIOUS WORK

In 2000, Muthukrishnan and Stroud [1] developed multi-valued logic for multi-level quantum computing systems and showed their realizability in linear ion trap devices. However, this approach produces circuits that are too large and no procedure was proposed to minimize them. In 2002, Brylinski and Brylinski [2] discussed the universality of n-qudit gates without giving any design algorithm Since 2001, Al-Rabadi and Perkowski [3],[4], and Khan et al [5],[6] proposed Galois Field approach to multi-valued quantum logic synthesis in several regular structures. They used gates that are ternary counterparts of classical binary Feynman and Toffoli gates, but no experimental data were given. De Vos [7] proposed two ternary 1*1 gates and two ternary 2*2 gates, but again no synthesis method was proposed. In 2002, Perkowski, Al-Rabadi, and Kerntopf [8] proposed a 2*2 Generalized Ternary Gate (GTG gate) based on the ternary conditional gate [1] and ternary shift gates [5],[6] and showed the realization of ternary Toffoli gate using GTG gates. This work introduced for the first time the practical realizability of Galois Field circuits in existing multi-valued quantum technology. Unfortunately, very little has been published on synthesis algorithms for multi-valued quantum circuits. More importantly, there is nothing published on synthesizing incompletely specified multi-output circuits, which is the problem dealt with in this paper.

III. FUNDAMENTALS OF MULTI-VALUED QUANTUM LOGIC

In multi-valued (MV) Quantum Computing (QC), the unit of memory (information) is ***qudit***. MV quantum logic operations manipulate qudits, which are microscopic entities such as a photon's polarization or atomic spin. Ternary logic values of 0, 1, and 2 are represented by a set of distinguishable different states of a ***qutrit***. These states can be a photon's polarizations or an elementary particle's spins. After encoding these distinguishable quantities into multiple-valued constants, qutrit states are represented by $|0\rangle$, $|1\rangle$, and $|2\rangle$, respectively.

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

Pairs of qutrits are capable of representing nine distinct states, $|00\rangle$, $|01\rangle$, $|02\rangle$, $|10\rangle$, $|11\rangle$, $|12\rangle$, $|20\rangle$, $|21\rangle$, and $|22\rangle$, as well as all possible superpositions of the states. This property may be mathematically described using the Kronecker product (tensor product) operation ⊗ [9]. The Kronecker product of matrices is defined as follows:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \otimes \begin{bmatrix} x & y \\ z & v \end{bmatrix} = \begin{bmatrix} a\begin{bmatrix} x & y \\ z & v \end{bmatrix} & b\begin{bmatrix} x & y \\ z & v \end{bmatrix} \\ c\begin{bmatrix} x & y \\ z & v \end{bmatrix} & d\begin{bmatrix} x & y \\ z & v \end{bmatrix} \end{bmatrix} = \begin{bmatrix} ax & ay & bx & by \\ az & av & bz & bv \\ cx & cy & dx & dy \\ cz & cv & dz & dv \end{bmatrix}$$

As an example, consider two qutrits with $\psi_1 = \alpha_1|0\rangle + \beta_1|1\rangle + \gamma_1|2\rangle$ and $\psi_2 = \alpha_2|0\rangle + \beta_2|1\rangle + \gamma_2|2\rangle$. When the two qutrits are considered to represent a state, that state ψ_{12} is the superposition of all possible combinations of the original qutrits, where

$\psi_{12} = \psi_1 \otimes \psi_2 = \alpha_1\alpha_2|00\rangle + \alpha_1\beta_2|01\rangle + \alpha_1\gamma_2|02\rangle + \beta_1\alpha_2|10\rangle + \beta_1\beta_2|11\rangle + \beta_1\gamma_2|12\rangle + \gamma_1\alpha_2|20\rangle + \gamma_1\beta_2|21\rangle + \gamma_1\gamma_2|22\rangle$

Superposition property allows qubit states to grow much faster in dimension than classical bits, and qudits faster than qubits [1]. In a classical system, *n* bits represent 2^n distinct states, whereas *n* qutrits correspond to a **superposition** of 3^n states. In the above formula some coefficient can be equal to zero, so there exist a constraint bounding the possible states in which the system can exist. As observed in [1] – "*Allowing d to be arbitrary enables a tradeoff between the number of qudits making up the quantum computer and the number of levels in each qudit*". An output of a gate is obtained by multiplying the unitary matrix of this gate by the vector of Hilbert space corresponding to this gate's input state. A resultant unitary matrix of arbitrary quantum circuit is created by matrix or Kronecker multiplications of matrices of composed subcircuits. These all contribute to difficulty in understanding the concepts of quantum computing and creating efficient analysis, simulation, verification and synthesis algorithms for QC. Generally, however, we believe that much can be learned from the history of Electronic Computer Aided Design as well as from MV logic theory and design, and the lessons learned there should be used to design efficient CAD tools for MV quantum computing.

In terms of logic operations, anything that changes a vector of qudit states to another qudit satisfying measurement probability properties can be considered as a quantum operator (unitary matrix). These phenomena can be modeled using the analogy of a "quantum circuit". In a quantum circuit, wires do not carry ternary constants but correspond to 3-tuples of complex values, α, β, and γ. Quantum logic gates of the circuit map the complex values on their inputs to complex values on their outputs. As mentioned, operation of quantum gates is described by matrix operations. Any quantum circuit is a composition of parallel and serial connections of blocks, from small to large. Small blocks correspond to directly realizable quantum gates such as Feynman or Stroud/Muthukrishnan gates. Serial connection of blocks corresponds to multiplication of their (unitary) matrices. Parallel connection corresponds to Kronecker multiplication of their matrices. So, theoretically, the analysis, simulation and verification are easy and can be based on matrix methods. Practically they are tough because the dimensions of the matrices grow exponentially. All these become much easier when one deals only with permutative matrices, which are equivalent to multi-output truth tables of completely specified functions. We deal with such a special class in this paper.

IV. SOME TERNARY PERMUTATION QUANTUM GATES

Any unitary matrix represents a quantum gate. If a unitary matrix has only one 1 in every column and the remaining elements are 0, then such a matrix is called a ***permutation matrix***. A quantum gate represented by a permutation matrix is called a ***permutation quantum gate***. In this paper we concentrate only on permutation quantum gates.

Figure 1 shows a 2*2 ternary ***Feynman gate***. Here *A* is the controlling input and *B* is the controlled input. The output *P* is equal to the input *A* and the output *Q* is GF3 sum of *A* and *B*. Observe that GF3 sum is the same as modulo 3 sum. If $B = 0$, then $Q = A$ and the ternary Feynman gate acts as a ***copying gate***. The ternary 2*2 Feynman gate is practically realizable, for instance see [1].

Six 1*1 ***ternary shift gates*** are proposed in [5, 6]. Operations

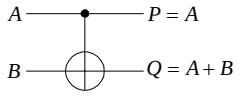

Figure 1. 2*2 ternary Feynman gate.

Gate Name	Gate Symbol with operaton *	Gate No. to refer in the text
Buffer	x — x	0
Single-Shift	x — $x' = x+1$	1
Dual-Shift	x — $x'' = x+2$	2
Self-Shift	x — $x''' = 2x$	3
Self-Single-Shift	x — $x^{\#} = 2x+1$	4
Self-Dual-Shift	x — $x^{\wedge} = 2x+2$	5

* Addition and multiplication are over GF3.

Figure 2. Ternary shift gates.

and symbols of these gates are shown in Fig. 2. These gates are realizable using ternary quantum Feynman primitive [5, 6].

A shift gate is said to be a ***mirror gate*** of another shift gate if the mirror gate is connected with the output of the original shift gate, then the input signal is restored. The mirror gates for all the shift gates are shown in Figure 3.

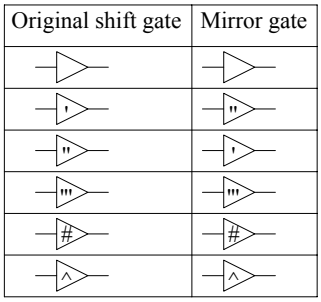

Figure 3. Mirror gates.

A very useful 2*2 gate called ***Generalized Ternary gate (GTG gate)*** is proposed in [8] as shown in Figure 4. Here, input *A* is the controlling input and input *B* is the controlled input. The output *P* is equal to the input *A*. The controlling input *A* controls a conceptual ternary multiplexer (a conditional gate) that can be realized using quantum technology such as ion traps [1]. If $A = 0$, then the output *Q* is the *x* shift of the input *B*. Similarly, if $A = 1$, then the output *Q* is the *y* shift of the input *B* and if $A = 2$, then the output *Q* is the *z* shift of the input *A*. Here shift means all ternary shift operations including the Buffer (simple quantum wire). As the Conditional gate and the Shift gates are realizable in quantum technology, the GTG gate is a truly realizable ternary quantum gate.

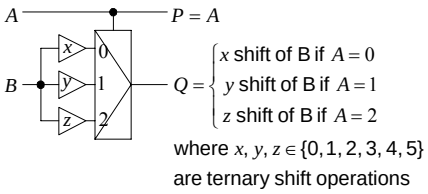

Figure 4. 2*2 Generalized Ternary gate.

For the purpose of this paper we assume that the GTG gate can be controlled from both top and bottom as shown in Figure 5.

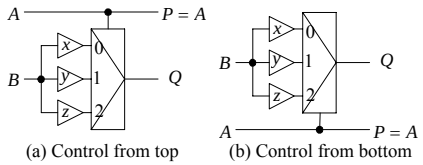

Figure 5. Two different forms of controlling a GTG gate.

It should be noted that if $x = y = z = 0$, then for all values of *A*, $Q = B$ and the GTG gate eventually becomes equivalent to two parallel wires as shown in Figure 6(a). Again, if $A = 0$ and $x = 0$ as in Figure 6(b); if $A = 1$ and $y = 0$ as in Figure 6(c); and if $A = 2$ and $z = 0$ as in Figure 6(d), then the GTG gate also becomes equivalent to two parallel wires.

A very useful gate for multiple input circuit synthesis is a 3*3 ***Toffoli gate*** as shown in Figure 7. Design of GFSOP (Galois Field Sum of Products) arrays and factorized arrays is based on these gates. These arrays are the multiple-valued counterparts of well-known binary ESOP (Exclusive Sum of Products) and factorized ESOP cascades. Here the inputs *A* and *B* are the controlling inputs and the input *C* is the controlled input. The output *P* is equal to the input *A*, the output *Q* is equal to the input *B*, and the output *R* is equal to $A \bullet B + C$, where $\bullet$ and + are GF3 multiplication and addition, respectively.

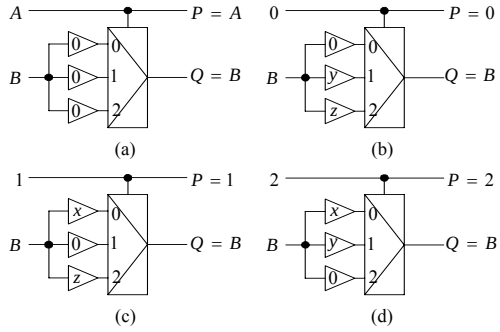

Figure 6. Some configurations of GTG gate that act as two parallel wires.

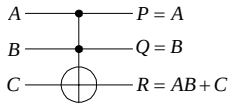

Figure 7. 3*3 Toffoli gate.

A generalized Ternary Toffoli gate is proposed in [6] as shown in Figure 8, where f_k is an arbitrary ternary function of the input variables $A_1, A_2, \cdots, A_k$. Here, there are *k* controlling inputs and *n* controlled inputs.

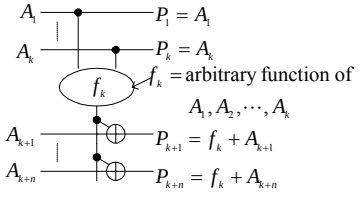

Figure 8. A generalized ternary Toffoli gate.

Any *m***m* (*m* > 2) gate is very difficult to realize in quantum technology, since interaction of more than two particles is nearly impossible to control. Therefore, these gates should be

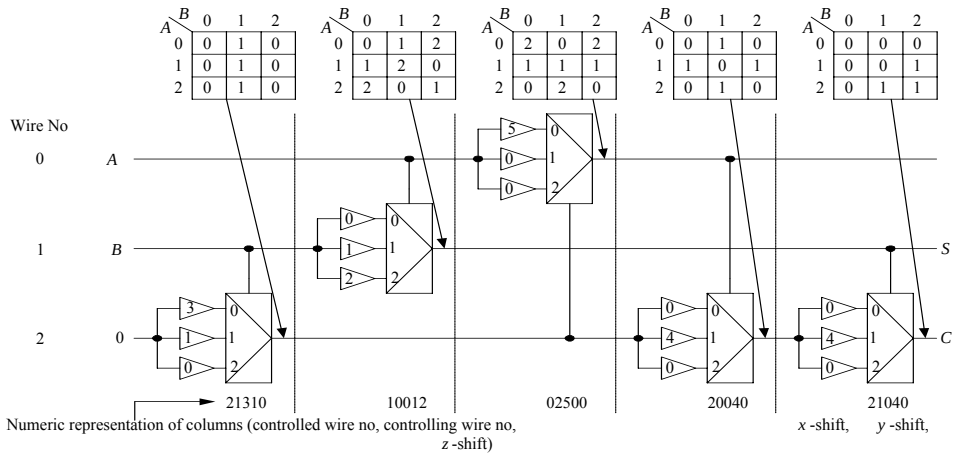

Figure 10. Realization of ternary half-adder function $C(A,B) = [0, 0, 0, 0, 0, 1, 0, 1, 1]^T$ and $S(A,B) = [0, 1, 2, 1, 2, 0, 2, 0, 1]^T$ using a cascade of GTG gates.

	P_C	0.6		0.7		0.8		0.9		1.0	
P	L P_M	1/L	2/L	1/L	2/L	1/L	2/L	1/L	2/L	1/L	2/L
50	9										
	18		8, 4	8, 4	7, 4	8, 4			7, 4	6, 4	
	27	11, 5	15, 5	6, 4	8, 5	12, 4		12, 5	15, 4	11, 4	10, 4
100	9										
	18	7, 4				9, 5		7, 4		8, 3	9, 4
	27	12, 4	**5, 3**		14, 5	19, 5	11, 4			7, 4	
150	9										
	18		6, 3	8, 4				5, 4	6, 3	7, 4	
	27	10, 5	12, 4		15, 5	11, 5		9, 4	17, 5	8, 4	8, 4
200	9				5, 4						
	18			6, 4	8, 5	**5, 3**	6, 3				
	27	17, 5	12, 5	15, 5	8, 5	11, 5	16, 5		9, 4	9, 4	6, 4
250	9										
	18	9, 5		7, 3						8, 4	6, 5
	27	9, 5	8, 4	7, 3	8, 5		14, 5	16, 5	8, 4	9, 4	6, 4
300	9										
	18	6, 4	7, 4	7, 4	8, 4		7, 4	7, 4	7, 3		
	27		14, 5			8, 3	14, 4	10, 4	16, 5	14, 5	7, 4
350	9										
	18	9, 5	5, 4	7, 4	6, 4	7, 5	7, 5	6, 4	8, 5	8, 4	
	27	7, 4	12, 5	10, 4	10, 5	12, 5	10, 4		12, 5		9, 4
400	9										
	18	6, 4	5, 4	8, 4	11, 5		5, 4	7, 4		11, 5	6, 4
	27	17, 5			8, 4	7, 4	13, 5	10, 5	16, 5	7, 4	
450	9								6, 4		
	18	8, 5	7, 5	6, 4	10, 4			8, 5	9, 5		8, 4
	27	7, 5	12, 5	6, 3	15, 5	16, 5	8, 4	7, 3	14, 5		14, 5
500	9							**5, 3**			
	18	8, 5	6, 5	12, 5			6, 4	11, 5		10, 4	7, 4
	27	16, 5		**5, 3**	6, 4	12, 5	13, 4	8, 4	10, 5	13, 5	7, 4

Table 1. Number of GTG gates and scratchpad register width (separated by commas) for ternary half-adder function generated by the GA for different values of population size (P), chromosome length (L), crossover probability (PC), and mutation probability (PM). An empty entry represents that the GA did not find a correct circuit within 500,000 generations.

realized from 1*1 and 2*2 gates. As the ternary Feynman gate and GTG gate are relatively easy to realize, they are treated as primitive gates for realizing other gates. A generalized Toffoli gate is realizable from 1*1 Shift gates, 2*2 Feynman gate, and 2*2 GTG gate as discussed in Section IX. Quantum technologies do not allow wire crossing. In those technologies, swap gate plays an important role. (In general reversible logic these gates may have zero cost since any two wires can overlap). The schematic of a ternary swap gate is shown in Figure 9.

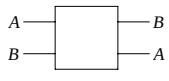

Figure 9. Ternary swap gate.

V. GENERAL MODEL OF SYNTHESIZING MULTI-OUTPUT TERNARY FUNCTIONS USING CASCADES OF GTG GATES

Realization of ternary half-adder function $C(A,B) = [0, 0, 0, 0, 0, 1, 0, 1, 1]^T$ and $S(A,B) = [0, 1, 2, 1, 2, 0, 2, 0, 1]^T$ using a cascade of GTG gates is shown in Figure 10. Signal values at all intermediate wires are shown as maps to verify the correctness of the circuit. In this realization we assumed the following:

(1) A GTG gate can be controlled either from top or from bottom.
(2) A limited vertical wire crossing for the controlling signals of GTG gates is allowed.
(3) Constant input signals 0, 1, or 2 are added as needed.
(4) Output may be realized along any primary input line or any constant input line.
(5) Each of the GTG gate form a column where the remaining lines represent quantum wires. The columns are cascaded to realize the circuit.

VI. PROPOSED GENETIC ALGORITHM

A. Problem encoding

In the proposed genetic algorithm (GA) we use the model of synthesizing multi-output ternary function using cascaded GTG gates as discussed in Section IV. In this circuit model, for initial input to the GA, we add three constant input signals 0, 1, and 2 for up to $m \le n+3$ outputs, where n is the number of inputs. For every increment of 3 or less outputs, we add additional 3 constant input signals 0, 1, and 2. For example, if the function has 2 inputs and 6 outputs, then we add 6 constant input signals 0, 1, 2, 0, 1, and 2. Then after convergence of the GA we eliminate the unused constant input signals from the final circuit. Initially we take 3^n or 2×3^n or 3×3^n columns (chromosome length) as the input to the GA. After convergence of the GA we eliminate a column having all wires (i. e. a column having a GTG gate representing two parallel wires) and other redundant columns as described in Subsection VI. F.

The primary input lines and the constant input lines are numbered starting from 0 as shown in Figure 10. Each of the columns of the circuit is represented by an ordered tuple of controlled wire no, controlling wire no, *x*-shift, *y*-shift, and *z*-shift of the associated GTG gate as shown in Figure 10. Using this notation the chromosome representing the circuit of Figure 10 is as shown in Figure 11 (these are strings of characters and not integers). Here each column of the circuit is a gene of the chromosome. In this problem encoding of the genotype (chromosome) ties very closely with the phenotype (actual circuit).

21310	10012	02500	20040	21040

Figure 11. Chromosome representing the circuit of Figure 10.

B. Fitness function

In the proposed GA, we tried to reduce the cost of the resulting circuit by (i) reducing the number of wires in the circuit (the width of the scratchpad register), i. e. increasing the number of unused constant input lines, (ii) reducing the number of non-wire columns, i. e. increasing the number of wire columns, (iii) reducing the number of non-buffer shift gates, i. e. increasing the buffer gates in the non-wire columns. For this reason we used four components of the fitness function as discussed below.

Output fitness: The output fitness is defined as follows:

Individual output fitness, O_i = (if output i is realized along any wire, then 1, otherwise 0) + highest number of truth values realized along any wire/ 3^n

Total output fitness, $O = \sum_{i=1}^{m} O_i$, where m is the number of outputs in the function.

For testing the output fitness, we compute the resulting truth vector for all wires and then the best fit wire is selected for a given output i.

Width fitness: The scratchpad width fitness is defined as follows:

W = Number of unused constant input lines/Number of constant input lines.

Column fitness: The column fitness (or cascade length fitness) is defined as follows:

C = Number of wire columns/length of the chromosome, L.

Shift-gate fitness: The shift-gate fitness is defined as follows:

S = Number of buffer gates in the non-wire columns/3×Number of non-wire columns.

In the current quantum technologies the scratchpad width is a major limitation. Therefore, if we can reduce the width of the circuit, it will be more favorable. So, we give more selection pressure on width fitness. Reducing the number of columns will reduce the cost of the circuit. So, we give moderate selection pressure on column fitness. Finally, reducing the number of non-buffer shift gates also reduce the cost of the circuit to some extent. So, we give less selection pressure on shift-gate fitness. Considering all these factors, we define the fitness function as follows:

$$F = O + 0.5W + 0.4C + 0.1S$$

From the fitness function, we can see that the value of $0.5W + 0.4C + 0.1S$ will always be less than 1. On the other hand, when all the m outputs are realized, then the value of O will be $2m$. Therefore, the threshold fitness value is $2m$, that is, if the fitness of a chromosome is greater than $2m$, then that chromosome is a solution for the given function.

C. Type of GA

As the model of our circuit synthesis (see Section V) is not well structured, we want to make sure that the best solutions found are not lost in the successive generations. Therefore, we use the steady-state GA [12]. We use gene repair, binary tournament selection and elimination of redundant columns (knowledge-based local transformation).

D. GA operators and parameters

We experimented with different values of population size (P), chromosome length (L), crossover probability (P_C), and mutation probability (P_M) for synthesizing ternary half-adder and we have done replicate trials for each parameter settings. The influences of these parameters are shown in Table 1. From Table 1, we see that a wide range of population sizes yield good solutions. Therefore, in our other experimentation, we used population sizes of 100, 200, 300, 400, and 500.

In the experimentation of Table 1, we used chromosome length (number of columns) of 3^n, 2×3^n, and 3×3^n (that is, 9, 18, and 27). From the table, we see that chromosome length of 2×3^n and 3×3^n yield good solutions. Therefore, in our other experimentation, we used chromosome length of 2×3^n and 3×3^n.

In our GA we use binary tournament selection with replacement for selecting the parents. One-point crossover was used and, as shown in Table 1, crossover probabilities of 0.6, 0.7, 0.8, 0.9, and 1.0 all yielded good results.

We mutated each column (or gene) of the offspring with a given low mutation probability (P_M). In this mutation we replaced the column by a randomly generated column. In our experimentation illustrated in Table 1, we used mutation probability of 1/*L* and 2/*L*, where *L* is the chromosome length, and we see from table that both of these two mutation probabilities yield good results. Our GA seemed to be not much sensitive to crossover and mutation probabilities, so we concentrated on repair which had big influence on results quality. However, further studies need to be done on influence of various parameters and other genetic operators.

E. Repair operation

From the circuit model of Figure 10, we see that, in the gene representation of a column, the wire numbers representing the controlled signal and the controlling signal should be different. But if, during random generation of the individuals of the initial population or after mutation of offspring, both the wire numbers of a gene become same, then we make that column representing wires by setting $x = y = z = 0$. The motivation behind this repair operation is to reduce the number of non-wire columns in the final solution. As our circuit model initially starts with an arbitrary length, reducing the number of non-wire columns will improve the quality of the solution. For example, if a gene is 11012, then we make it 11000.

F. Elimination of redundant columns

In the solution produced by the GA, some of the columns will be wire-columns. We eliminate all such wire-columns from the solution to get the final solution. But, even after elimination of these wire-columns some of the remaining columns may still be redundant. For example, a GA may produce (after the wire-columns have been eliminated) the circuit of Figure 12 for ternary half-adder function. The third and the sixth columns from the left are redundant, because they modify the garbage outputs [13]. Therefore, we also eliminate these redundant columns from the solution to get the final solution.

For a given function, we performed a number of experiments using different values of population size (*P*), chromosome length (*L*), crossover probability (P_C), and mutation probability (P_M) as stated above. We eliminated redundant columns from all these solutions. Then we selected the best solution from these experiments as the final solution for the given function. The circuit of Figure 10 is thus derived for ternary half-adder function.

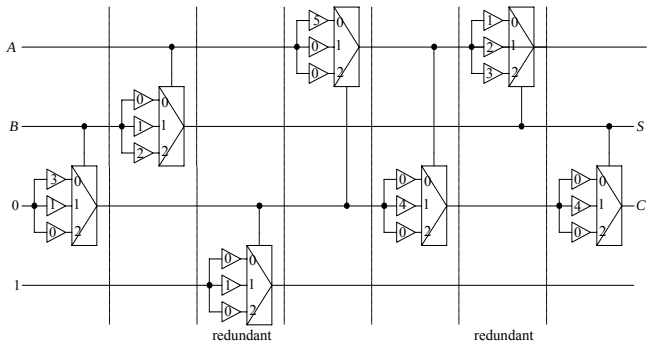

Figure 12. Ternary half-adder circuit with redundant columns.

VII. SYNTHESIS OF INCOMPLETELY SPECIFIED MULTI-OUTPUT TERNARY FUNCTIONS

For synthesizing an incompletely specified multi-output ternary function, we used the same GA as discussed in Section VI, except the output fitness is calculated differently because don't cares are ignored. Now, the truth vectors of a wire and the output function are compared only for cares. Interestingly, this allows to simplify functions with more wires faster, when the percent of don't cares is high. We experimented with a randomly generated 2-input 3-output incompletely specified function $F_0(A,B) = [3, 2, 1, 3, 1, 3, 2, 0, 1]^T$, $F_1(A,B) = [1, 2, 3, 2, 3, 3, 2, 0, 2]^T$, and $F_2(A,B) = [0, 0, 3, 0, 3, 1, 2, 2, 2]^T$, where a 3 represents a don't care output. The resulting circuit is shown in Figure 13. In the figure, intermediate signal values are shown as maps to verify the correctness of the circuit.

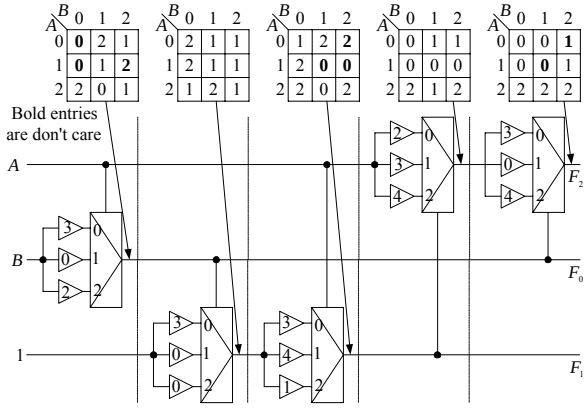

Figure 13. Circuit realizing an incompletely specified function $F_0(A,B) = [3, 2, 1, 3, 1, 3, 2, 0, 1]^T$, $F_1(A,B) = [1, 2, 3, 2, 3, 3, 2, 0, 2]^T$, and $F_2(A,B) = [0, 0, 3, 0, 3, 1, 2, 2, 2]^T$.

VIII. EXPERIMENTAL RESULTS

We have written C program to implement the proposed GA. We performed experimentation with some multi-output completely and incompletely specified ternary functions and the results are given in Table 2. Using the algorithm we were able to find better solutions to some known circuits and the program "discovered" new realizations of some known gates such as swap which had very inefficient realizations [5],[6],[8]. Synthesis of gates such as Toffoli and Swap is especially important because of the role that they play in other methods.

For synthesis of generalized ternary Toffoli gate, we assumed that the controlling function of Figure 8 is $f_k = A_1 A_2 \cdots A_k$. This type of generalized ternary Toffoli gates are very useful for realization of GFSOP cascades. For synthesizing a generalized ternary Toffoli gate of this type, we first realize the controlling function $f_k = A_1 A_2 \cdots A_k$ as a cascade of GTG gates using our GA and then the controlled outputs are realized using ternary Feynman gates. To restore the primary inputs and the constant inputs mirror GTG gates are used. Here we use the same GA as discussed in Section VI, except that the controlling function $f_k = A_1 A_2 \cdots A_k$ is realized along any constant input wires. We experimented with a generalized ternary Toffoli gate with 2 controlling inputs and 2 controlled inputs. The resulting circuit is shown in Figure 14. In this figure, the left four columns generate the controlling function AB along the constant input signal 2 and the right four columns are the mirror columns that restores the controlling inputs *A* and *B* and the constant input 2. Intermediate signal values are shown as maps to verify the correctness of the circuit. We synthesized ternary swap gate using cascade of GTG gates using our GA as discussed in Section 6, except that no constant input is used and the outputs are restricted to their corresponding wires. The resultant circuit is shown in Fig. 15.

IX. CONCLUSIONS

GTG gate was proposed in [8] without giving any synthesis algorithm. In this paper we prpose a GA for synthesizing both completely and incompletely specified ternary functions using cascade of GTG gates. Generalized ternary Toffoli gate and ternary swap gate were synthesized. The generalized ternary Toffoli gate realization proposed in [8] requires 10 GTG gates, whereas the realization of this paper requires 8 GTG gates. Similarly, previous best design of ternary swap gate had 4 Feynman gates and one 1-qubit permutative gate. The new design has only 3 GTG gates and is very elegant, it has the same symmetry as the well-known design of Swap from Feynman gates in binary, so we can say that the GA has done certain "discovery". Other circuits are realized using cascade of GTG gates for the first time and, therefore, cannot be compared with other results.

X. FUTURE WORK

Future research is further improvement of the GA to a broader class of evolutionary algorithms (larger tournaments, restart with new parameters when stacked in local minimum, new crossover and mutation operators, local search [13], memetic algorithms). We will be also developing GA for synthesizing both completely and incompletely specified multi-output ternary function using cascade of both 2*2 GTG gates and 2*2 ternary Feynman gates. (Feynman gates are linear, although Feynman gate is a special case of GTG gate, it is treated in a special way as seen in rows 7,8 of Table 2). Similarly as in [13], we will add powerful local transformations of circuits based on ternary quantum identities, to decrease the cost of the synthesized cascades. In binary quantum the improvements of costs are sometimes as dramatic as 300% [13], which demonstrates that it is a good idea to combine evolutionary and algorithmic rule-based approaches into one working program for quantum circuits synthesis.

REFERENCES

[1] A. Muthukrishnan, and C. R. Stroud, Jr., "Multivalued logic gates for quantum computation", *Phys. Rev. A,* Vol. 62, No. 5, Nov. 2000, 052309/1-8.

[2] J. L. Brylinski and R. Brylinski, "Universal Quantum Gates", *Math. of Quantum Comp.*, CRC Press, 2002, LANL e-print quant-ph/010862.

[3] A. Al-Rabadi, "Novel Methods for Reversible Logic Synthesis and Their Application to Quantum Computing", *Ph. D. Thesis*, PSU, Portland, Oregon, USA, October 24, 2002.

[4] A. Al-Rabadi and M. Perkowski, "Multiple-Valued Galois Field S/D Trees for GFSOP Minimization and their Complexity", *Proc. 31st ISMVL,* Warsaw, Poland, May 22-24, 2001, pp. 159-166.

[5] M.H.A. Khan, M.A. Perkowski, and P. Kerntopf, "Multi-Output Galois Field Sum of Products Synthesis with New Quantum Cascades", *Proc. 33rd ISMVL,* Tokyo, May 16-19, 2003, pp. 146-153

[6] M.H.A. Khan, M.A. Perkowski, M.R. Khan, and P. Kerntopf, "Ternary GFSOP Minimization using Kronecker Decision Diagrams and Their Synthesis with Quantum Cascades", Submitted to *J. Multiple-Valued Logic and Soft Computing.*

[7] A. De Vos, B. Raa, and L. Storme, "Generating the group of reversible logic gates", *J. Physics A: Mathematical and General*, Vol. 35, 2002, pp. 7063-7078.

[8] M. Perkowski, A. Al-Rabadi, and P. Kerntopf, "Multiple-Valued Quantum Logic Synthesis", *Proc. of 2002 Int. Symposium on New Paradigm VLSI Computing,* pp. 41-47. Sendai, Japan, December 12-14, 2002

[9] Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 2000.

[10] M. Hirvensalo, "Quantum Computing," *Springer Verlag*, 2001.

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

[12] P. Mazumder and E.M. Rudnick, *Genetic Algorithms for VLSI Design, Layout & Test Automation*, Pearson Education Asia, 2002.

[13] M. Lukac, M. Perkowski, H. Goi, M. Pivtoraiko, C.H. Yu, K. Chung, H. Jee, B-G. Kim, and Y-D. Kim, "Evolutionary approach to Quantum and Reversible Circuits synthesis", Artificial Intelligence Review, 20, pp 361-417, 2003. Kluwer Academic Publishers.

[14] 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. Springer Verlag, London, 1998.

[15] C.W. Williams, A.G. Gray, "Automated Design of Quantum Circuits", ETC Quantum Computing and Quantum Communication, *QCQC '98,* Palm Springs, California, February 17-20, *Springer-Verlag*, pp. 113-125, 1999.

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

[17] 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.

[18] L. Spector, H. Barnum, H.J. Bernstein, and N.Swamy, "Finding a better-than-classical quantum AND/OR algorithm using genetic programming," *Proc. 1999 Congress on Evolutionary Computation,* Vol. 3, pp. 2239-2246, Washington DC, 6-9 July 1999, IEEE, Piscataway, NJ.

[19] K-H Han, J-H Kim, Quantum-inspired evolutionary algorithm for a class of combinatorial optimization, *IEEE Trans. Evolutionary Computation, 6 (6)*: pp. 580-593, 2002.

Table 2. Number of GTG gates, scratchpad width, and number of additional Feynman gates required for realizing some multi-output ternary functions.

Function	In	Out	No. of GTG gates	Scratchpad width	No. of additional Feynman gates
thadd (ternary half-adder): $C(A,B)=\text{int}[(A+B)/3]$, $S(A,B)=(A+B) \bmod 3$	2	2	5	3	0
mul2: $C(A,B)=\text{int}[AB/3]$, $M(A,B)=AB \bmod 3$	2	2	8	4	0
sum2: $F(A,B)=(A+B) \bmod 3$	2	1	2	3	0
sum3: $F(A,B,C)=(A+B+C) \bmod 3$	3	1	2	3	0
sqsum2: $F(A,B)=(A^2+B^2) \bmod 3$	2	1	2	3	0
avg2: $F(A,B)=\text{int}[(A+B)/2] \bmod 3$	2	1	3	3	0
gttg22 (generalized ternary Toffoli gate): $P=A$, $Q=B$, $R=AB+C$, $S=AB+D$	4	4	8	5	2
gttg33 (generalized ternary Toffoli) gate: $P=A$, $Q=B$, $R=C$, $S=ABC+D$, $T=ABC+E$, $U=ABC+F$	6	6	30	8	3
tsg (ternary swap gate): $P=B$, $Q=A$	2	2	3	2	0
Randomly generated incompletely specified function: $F_0(A,B)=[3,2,1,3,1,3,2,0,1]^T$, $F_1(A,B)=[1,2,3,2,3,3,2,0,2]^T$, $F_2(A,B)=[0,0,3,0,3,1,2,2,2]^T$	2	3	5	3	0

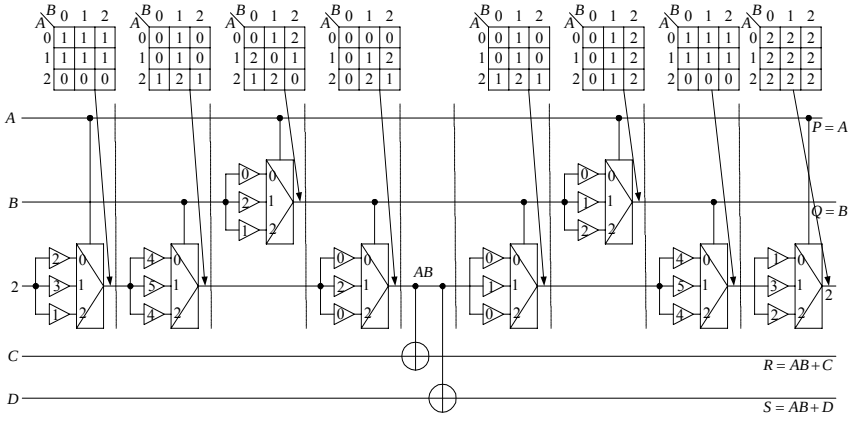

Fig. 14. Generalized Ternary Toffoli.

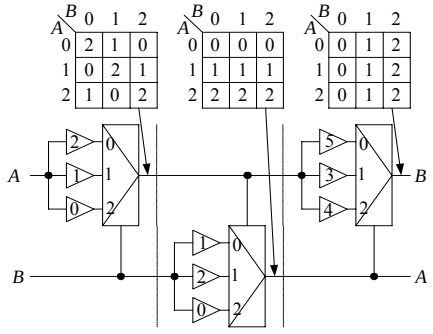

Fig.15. Realization of Ternary Swap gate