Evolutionary approach to Quantum and Reversible Circuits synthesis

Martin Lukac, Marek Perkowski, Hilton Goi, Mikhail Pivtoraiko[+], Chung Hyo Yu, Kyusik Chung, Hyunkoo Jee, Byung-guk Kim, Yong-Duk Kim

Department of Electrical Engineering and Computer Science,
Korea Advanced Institute of Science and Technology,
373-1 Guseong-dong,Yuseong-gu, Daejeon 305-701, Korea,

+Department of Electrical Engineering, Portland State University,
Portland, Oregon, 97207-0751, USA

lukacm@ece.pdx.edu
mperkows@ee.kaist.ac.kr

Abstract: The paper discusses the evolutionary computation approach to the problem of optimal synthesis of Quantum and Reversible Logic circuits. Our approach uses standard Genetic Algorithm (GA) and its relative power as compared to previous approaches comes from the encoding and the formulation of the cost and fitness functions for quantum circuits synthesis. We analyze new operators and their role in synthesis and optimization processes. Cost and fitness functions for Reversible Circuit synthesis are introduced as well as local optimizing transformations. It is also shown that our approach can be used alternatively for synthesis of either reversible or quantum circuits without a major change in the algorithm. Results are illustrated on synthesized Margolus, Toffoli, Fredkin and other gates and Entanglement Circuits. This is for the first time that several variants of these gates have been automatically synthesized from quantum primitives.

1. Introduction

Quantum computing is a flourishing and very attractive research area [7, 46, 48]. Inheriting properties from Quantum Mechanics, it allows theoretically to build computers much more efficient than the existing ones. For instance, certain problems non solvable in polynomial time in classical domain can be solved in polynomial time in quantum domain. Similarly, the complexity of other problems can be reduced while transforming them into the Quantum domain [48]. Moreover, Quantum Circuits (QC) have an advantage of being able to perform massively parallel computations in one time step [7,46,48]. The motivation to develop automated CAD tools for quantum circuits becomes recently quite high because according to the results of [37,38,39,40,41] such computers can be physically build and the progress of these realizations is fast. It is already possible to perform quantum-mechanically simple operations with trapped ions or atoms. Simplified but complete quantum circuits were constructed using Nuclear Magnetic Resonance technology [48]. The state of the art in year 2002 is a 7 qubit quantum computer [45]. For this size of computer the problem of quantum circuit synthesis and optimization is not trivial and cannot be solved by hand so some kind of design automation becomes necessary. While quantum mechanics and quantum computing are established research areas, systematic design methods of quantum computers, and especially logic circuit design for such computers still remain only at the beginning stages of exploring available possibilities. It can be compared to the state of the art of logic design in 1940's when the first standard binary computers were built and the minimization algorithms like those from Quine and Shestakov started to appear. Currently, the works in quantum logic design are on: designing new universal gates and investigating their properties, creating methods of composing gates to circuits, and building elementary quantum circuits for practical applications, for instance arithmetics. New algebraic models for quantum logic and circuit design are also investigated. The Computer Aided Design of Quantum Circuits is even less developed and there exist only few programs to design such circuits automatically or with a limited user intervention. This paper is related to new approaches for CAD of Quantum Circuits, the new research and development area that we try to establish by our research [1,2,10,28,29,30,31,32,33,42,43,55,56,57,58].

So far, analytic and search approaches have been used in quantum logic synthesis. They are based on matrix decomposition, local circuit transformations, mapping from various types of decision diagrams, spectral approaches, and on adaptations of several EXOR logic, Reed-Muller, and other classical combinational circuit design methods. Our approach combines some of them but fundamentally belongs to the group of Evolutionary Algorithms. The approach has been also used to a similar problem of Reversible Circuit (RC) Logic synthesis; such circuits can be realized in CMOS, optical and nano-technologies [47].

Genetic Algorithms (GAs) are one of the well-known Evolutionary Algorithm problem solving approaches to Soft Computing [9]. Their use is very popular in problems with no identified structure and high level of noise, because: 1) a big problem space can be searched, 2) the size of this space can be moderated by parameters, 3) a variety of new solutions can be produced, and 4) with long enough time a circuit can be obtained that is close to the optimal one. These advantages make GAs useful in the initial phases of research and investigations of the design problem space. GAs are very good candidates to be used in logic synthesis of new types of circuits, investigating the usefulness of new gate types and new circuit structures. The special cases of Evolutionary Algorithms include: Genetic Algorithms, Evolutionary Programming, Evolution Strategies and Genetic Programming. So far, to the best of our knowledge, only two of the four Evolutionary Algorithm types have been applied to the QC or RC synthesis. Genetic programming was used to synthesize EPR (Einstein-Podolsky-Rosen) pairs of qubits [5], while a growing number of works uses a classical GA for QC and RC synthesis. For instance, [1,2,3,6,7,8,54] used GA to evolve quantum and reversible gates and circuits, like the teleportation circuits. A reversible circuit is one that has the same number of inputs and outputs and is a one-to-one mapping between vectors of input and output values. Such circuits are related to quantum circuits. An attempt at a general approach to encode both Quantum and Reversible Circuits was presented in [1,2]. It is known that every quantum circuit is reversible [7,46,48], so the researches on classical binary reversible synthesis and quantum synthesis share many ideas. The Reversible Logic (RL) circuits [14,15,20] are already technologically possible and have been implemented in CMOS technology [47]. Thus, some of our results below are applicable also to such circuits. As described later, most of gates with more than one input used in QC are derived and originate from RC. Although this work is concerned with one approach to automated synthesis, it is important to notice that a parallel research on new gates is also explored by [10,20,21,22,23,24,25,26,27,28,29,50,51,52]. The search of new Quantum gates (QG) and Reversible gates (RG) has two different aspects: invention and generalization. The invention of new gates is mainly aimed toward an optimization of a particular design in a specific technology or towards inventing new universal gates. The generalization approach is the use of already known gates and exending them to new gates of certain preferable properties. Usually there are also new particular synthesis methods that come together with proposed new gates [30,31,32,33,34,55,57,58].

This paper is organized as follows. Section 2 presents the minimal background in quantum circuit design necessary to understand the paper. Examples are used to illustrate the most important notions. Section 3 describes the general problem of synthesis of quantum circuits (called also quantum arrays) from primitive quantum gates, especially using evolutionary algorithms. Section 4 presents decomposition of gates to smaller primitives related to the cost of these gates. More realistic cost function for gates are one of main innovations of this paper. Section 5 presents our entire minimization methodology for quantum arrays synthesis. The sixth section describes local optimizing transformations used in the post-processing stage of GA. Section 7 presents our variant of GA and its settings for this work. Section 8 gives more details on the most important aspect: the fitness function design for GA and its role. Section 9 presents other aspects of the Genetic Algorithm that we applied. Section 10 describes experimental results and section 11 discusses issues and advantages of this approach as well as our current and future research and open questions. Section 12 concludes the paper.

2. Fundamentals of Quantum Logic

In quantum computation quantum bits (qubits) are used instead of classical binary bits to represent information. These information units are derived from the states of micro-particles such as photons, electrons or ions. These states are the basis states (basis vectors, eigenstates) of the computational quantum system. Assume an electron with two possible spin rotations: +1/2 and –1/2. Using Ket notation [12,13] these distinguishable states will be represented as |0> and |1>, respectively. Each particle in a quantum domain is represented by a wave function describing it as having both properties of a wave and of a particle as introduced by [11,12,13,16]. Based on these properties, quantum computation inherited the powerful concept of superposition of states. Assume a particle p_1 be represented by a wave function $\psi_1 = \alpha_1|0> + \beta_1|1>$. The coefficients α and β are complex numbers called the eigenvalues. They must be in general complex because only having complex values in wave functions allows to eliminate themselves in order to satisfy some experimentally observed properties of quantum world, such as for instance in the Two-Slit experiment [17]. It is not our goal here to explain such experiments or formulate fundaments of quantum mechanics. (The reader can find information in literature, especially [48]). Our goal is only to introduce the formal calculus of quantum mechanics in order to explain the basic concepts of our CAD algorithms and especially the genetic algorithm for quantum circuit synthesis. Similarly as an engineer who has no understanding of electronic circuits, but knows only Boolean Algebra, is able to develop logic circuits synthesis software, one with no understanding of quantum phenomena and physics will be able to develop quantum CAD if he will only learn some fundamental quantum notation and associated algebraic properties and transformations. To teach these to the Reader is one of the goals of this paper.

The interpretation of wavefunction ψ_1 is that $|\alpha_1|^2$ is the probability of the particle being measured in state |0> and $|\beta_1|^2$ is the probability of that particle being measured in state |1>. Thus, the measurement or observation process transforms the quantum world of complex wavefunctions to the macro-world of events that occur with standard probabilities. The superposition of states is represented by these facts: (1) each of these probabilities can be non-null, (2) $|\alpha_1|^2+|\beta_1|^2=1$, and (3) if another particle p_2 with a wavefunction $\psi_2 = \alpha_2|0> + \beta_2|1>$ is added to the system, then the resulting wavefunction will be $|\psi_1\psi_2> = \alpha_1\alpha_2|00> + \alpha_1\beta_2|01> + \beta_1\alpha_2|10> + \beta_1\beta_2|11>$. The system can be in any possible state of the wavefunction and will collapse to one of the eigenstates when measured [19]. The space of quantum computing is thus much larger than in classical computing, which causes that efficient algorithms for synthesis and analysis of quantum circuits are more difficult to develop. However, certain similarities exist which will be useful for us to explain the synthesis issues, especially to the readers with a digital design engineering background. The equations introduced above are the result of the Kronecker product [18] on matrices of elementary quantum gates that operate "in parallel". The Kronecker Product of Matrices is defined as follows:

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

Mathematically, it is the Kronecker Product operation that allows the quantum logical system to grow dimensionally much faster than classical logics. Observe that in a quantum system n qubits represent a superposition of 2^n states while in a classical system n bits represent only 2^n distinct states. Operations over a set of qubits are defined as matrix operations and map the physical states into the Hilbert space. [13,46,48]. The concept of Hilbert space will be used below only in the most elementary way necessary to understand the calculations. States of the wave function are eigenvalues of this space. Each matrix-operator represents a modification to the complex coefficients of the wave function. The new coefficients result in the modification of probabilities to find the system in a particular basic state. But we do not have to worry about classical probabilities in this paper since the operation of a quantum circuit is purely deterministic, so we will always deal with eigenvalues (complex probability) rather than standard probabilities in circuit design. Consequently a quantum gate will be a matrix having for input the vector of complex coefficients of the waveform and producing a vector of complex coefficients as the output. An illustrative example can be seen in equation 1.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} \alpha|0\rangle \\ \beta|1\rangle \end{bmatrix} = \begin{bmatrix} a\alpha|0\rangle + b\beta|1\rangle \\ c\alpha|0\rangle + d\beta|1\rangle \end{bmatrix} \qquad (1)$$

where a, b, c, and d are complex coefficients of the matrix indicating the (complex) probability to transit from one state to another, and $\alpha|0>$, $\beta|1>$ are the (complex) wavefunction coefficients to be propagated through the matrix-operator. Less formally, we can think about this matrix as a quantum equivalent of a classical gate such as AND which transforms its input states into output states. The designer of quantum algorithms has to deal with standard probabilities, but the designer of quantum circuits, which is our interest here, deals only with operations in quantum world because his input problem is described in such a way.

Assume j to be the square root of -1. Let us denote by U^+ a hermitian matrix of matrix U, which means the complex conjugate of the transposed matrix U (the matrix U is first transposed and next each of its complex numbers is replaced with its conjugate, thus a – jb replaces a +jb). We say that gate U is unitary when $U^*U^+ = I$, where I is an identity matrix. It can be shown that because the probabilities must be preserved at the output of the quantum gate, all matrices representing quantum gates are unitary. Thus every quantum gate, block and the entire circuit is described by a unitary matrix. Every quantum gate has therefore exactly one inverse matrix – quantum computing is reversible; quantum gate matrices represent logically reversible gates. Some of those gates are exactly the same as in classical reversible computing, which allows to use some results of binary reversible computing in quantum computing. While in general the coefficients in unitary matrices of quantum circuits are complex numbers, there are some special and important gates for which unitary matrices are just permutation matrices. (Let us recall that a permutation matrix has exactly one "1" in every row and in every column and all other entries are zeros – it describes therefore an input-output permutation of value vectors). The gates whose unitary matrices are permutation matrices are called permutation gates and they correspond to gates of classical reversible logic. There exist, however, other important gates whose unitary matrices are not permutation matrices. Rotational operators (gates) and gates such as Hadamard gate (denoted by H) and Square Root of Not Gate (denoted by V) belong to this second category which we will call "truly quantum primitives". These gates are responsible for superposition, entanglement and all peculiarities of quantum computing, although they may constitute only a small fraction of gates in a quantum circuit. A Hadamard gate is an example of a gate that has a unitary matrix which is not a permutation matrix. Here are some useful matrices:

$$Hadamard = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad Pauli-X = NOT = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Pauli-Y = \begin{bmatrix} 0 & -j \\ j & 0 \end{bmatrix},$$

$$Pauli-Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad Phase = \begin{bmatrix} 1 & 0 \\ 0 & j \end{bmatrix}, \quad V = \sqrt{NOT} = \frac{1+j}{2}\begin{bmatrix} 1 & -j \\ -j & 1 \end{bmatrix}$$

We will denote these gates by H, X, Y, Z, S and V, respectively. Only X is permutative. It can be easily checked by the reader that multiplying the matrix of gate V by itself produces the matrix of Pauli-X which is an inverter or NOT gate. The reader can find interesting identities by multiplying these matrices by themselves and also by their inverse matrices. Such identities in quantum logic are used to simplify the quantum circuits that come directly from evolutionary algorithms (section 6). They play a role analogous to Boolean algebra identities such as De Morgan used in classical logic synthesis.

Below we will pay special attention to circuits composed of only permutation gates (these are the so-called permutation circuits, because their unitary matrices are permutation matrices). An example of a permutation gate is the Feynman gate (called also CNOT gate or Quantum XOR). This gate can be described by the following logic expressions: (A,B) => (P, Q) = (A, A⊕B), see Figure 1a. This equation means that the output bit P is just the same as its input A, while on the output of the second wire Q the operation $A \oplus B$ is performed. This operation (EXOR of A and B) is a standard logic operation. Sometimes notation A’= P and B’= Q is used (Figure 1c,d,e). (In other papers notation A’=A, and B’=B is used to underlie the fact that the result occurs on the same quantum wire A or B). These expressions are also related to the quantum matrix. For instance, the permutation matrix and the equations from Figure 1c describe the Feynman operator from Figure 1a,b. The careful reader may verify that this is indeed a unitary matrix and that it satisfies the definition of a permutation matrix as well. Observe that each of its rows and columns corresponds to one of possible states of input and output values: 00, 01, 10, and 11. The binary states are encoded to natural numbers as follows: 00=0, 01=1, 10=2, 11=3. We will use natural numbers to address rows and columns in matrices. Let us observe that when A = 0 and B = 0, then A’=B’=0. Thus, input combination 00 is transformed to output combination 00. This is represented by a value of “1” at the intersection of row 0 and column 0 of the matrix. Similarly, input combination 11 is transformed to output combination 10, which is represented by a value of “1” at the intersection of row 3 and column 2 (Figure 1c). Another variant of Feynman gate is in Figure 1d,e.

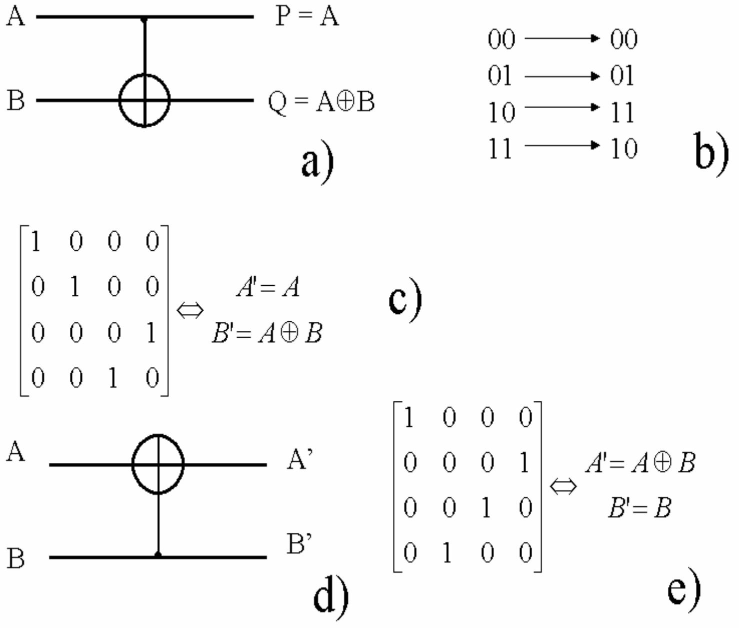


Figure 1: Feynman gates: (a) the circuit in quantum array notation, (b) the transformation between states executed by this gate, (c) the unitary matrix with binary enumeration of rows and columns and the corresponding Boolean equation of outputs, (d) Feynman EXOR up circuit scematics, (e) its unitary matrix and Boolean equations

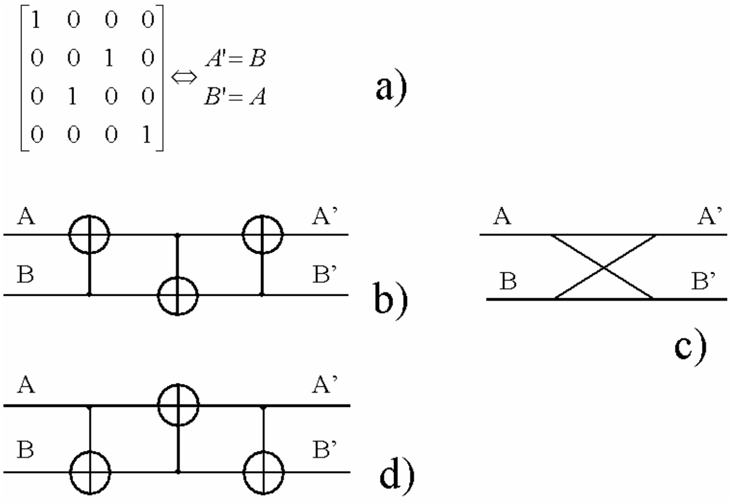


Figure 2: Swap gate: (a) unitary matrix and corresponding Boolean equations, (b) realization using Feynman gates, (c) a schematic, (d) another realization using two Feynman gates and one Feynman EXOR up gate.

Figure 2 presents another permutation gate called a Swap gate. Observe that this gate just swaps the wires A and B, since input state 01 is transformed to output combination 10, input vector 10 is transformed to output vector 01 and the remaining combinations are unchanged (Figure 2a). Swap gate is not necessary in classical CMOS reversible computing, where it is just a crossing of connecting wires that can be for instance in two layers of metallization. However, in quantum computing technology like NMR, every unitary matrix other than identity is realized by NMR electromagnetic pulses so its realization has a cost. High cost of swap gates in quantum realization is one of main differences between quantum and reversible circuit synthesis (in reversible computing these gates are free). Thus quantum circuit design covers some aspects of not only logic synthesis but also physical design (placement and routing) of standard CAD.

Let us introduce now few more gates that we will use. There are several 2*2 gates in binary reversible logic and they are all linear. A linear gate is one that all its outputs are linear functions of input variables. Let ***a, b***, and ***c*** be the inputs and ***P, Q*** and ***R*** the outputs of a gate. Assuming ***2*2 Feynman gate***, $\boldsymbol{P = a, Q = a \oplus b,}$ when $\boldsymbol{a = 0}$ then $\boldsymbol{Q = b;}$ when $\boldsymbol{a = 1}$ then $\boldsymbol{Q = \neg b.}$ (Sometimes the negation of a is also denoted by a'). With ***b=0 the 2*2*** Feynman gate is used as a ***fan-out (or copying) gate***. It can be shown that a swap gate can be built as a cascade of three Feynman gates – Figure 2 (the reader can check it by multiplying matrices or by transforming Boolean equations of gates). Figure 2b shows realization of Swap gate with three Feynman gates, and Figure 2c its schematics. Observe that there is also another realization of the Swap gate (Figure 2d). Circuits from Figure 2b and Figure 2d are then called equivalent or tautological. Every linear reversible function can be built by composing only 2*2 Feynman gates and inverters. There exist 8! = 40,320 3*3 reversible logic gates, some of them with interesting properties, but here we are not interested in types of gates but in synthesis methods using ***arbitrary*** permutation gates, so we will restrict ourselves to only few gate types (many other gate types have been defined in our software). There exist two well-known universal 3*3 reversible gates: Fredkin gate [14] and

Toffoli gate. Toffoli gate is also called 3*3 Feynman gate or Controlled-Controlled–NOT (Figure 3). The 3*3 Toffoli gate is described by these equations***:***

$P = a, \quad Q = b, \quad R = ab \oplus c.$

Toffoli gate is an example of ***two-through*** gates, because two of its inputs are given to the output. Similarly, the concept of ***k-through gates*** can be introduced, as well as the concept of k*k Toffoli Gates. In general, for a reversible gate with n inputs and n outputs, the matrix is of size $2^n * 2^n$.

The 3*3 *Fredkin gate* (Figure 4) is described by these equations:

P = a, Q = if a then c else b, R = if a then b else c.

As we see, in terms of classical logic this gate is just two multiplexers controlled in a flipped (permuted) way from the same control input ***a***. The symbol notation for a 3*3 Fredkin Gate is shown in Figure 4a. Qubit ***a*** is the controlling qubit. A classical schematics of this gate that uses multiplexers is shown in Figure 4b. As we see, the 3*3 Fredkin gates are permutation gates, they permute the data inputs ***b,c*** of the multiplexers under control of the control input ***a*** of these multiplexers that is also outputted from the gate. The fact that output ***P*** replicates input ***a*** is very useful because it allows to avoid fanout gates for the same level of the circuit (let us recall that fanout is not allowed in reversible logic). Copying of arbitrary signals is not allowed in quantum circuits (no cloning principle) [48], so the replication using Feynman gates can be applied only to basis states. Figure 4c presents a Fredkin gate controlled with qubit c. In Figure 4d qubit ***b*** is the controlling qubit. Realization of Fredkin gates using Toffoli and Feynman gates is shown in Figure 5.

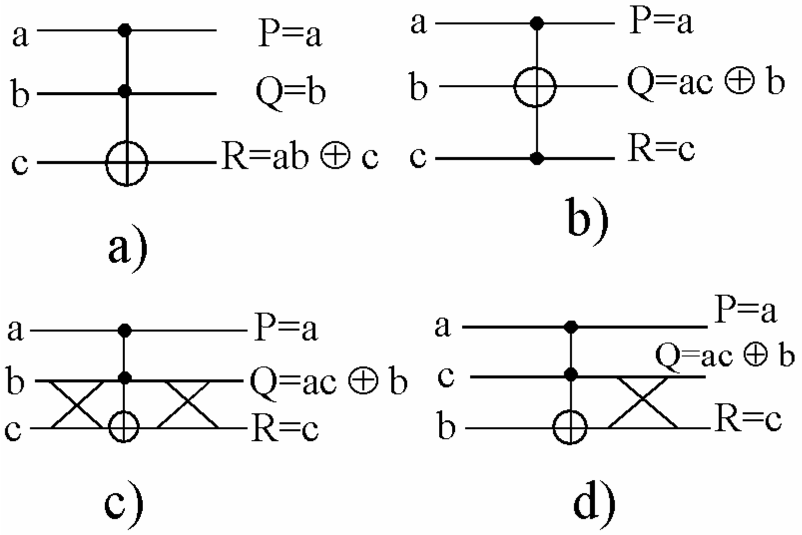


Figure 3. Toffoli gates: (a) a schematic of Toffoli gate (EXOR down), (b) Toffoli gate (EXOR middle), (c) realization of schematics from Figure 3b using the Toffoli EXOR down and two Swap gates, (d) realization of Toffoli EXOR middle with permuted input order

Fredkin gates are examples of what we define here as ***one-through gates***, which means gates in which one input variable is also an output.

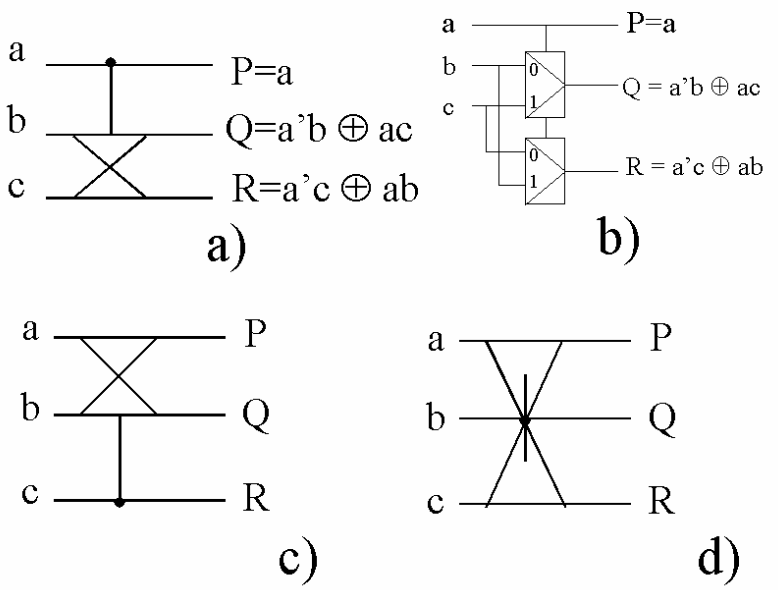


Figure 4. Fredkin gates: (a) Schematics with controlled qubit a, (b) the classical schematics of this gate using multiplexers, (c) Fredkin gate controlled by qubit c (Fredkin Down), (d) Fredkin gate controlled by qubit b (Fredkin middle).

Figure 6 presents the well-known realization of Toffoli gate from [21]. According to the figure in right (Figure 6b), there are five 2-qubit primitives. Let us explain how the quantum primitives cooperate to create correct gate behavior. The non-permutation matrices of quantum primitives are composed to create permutation

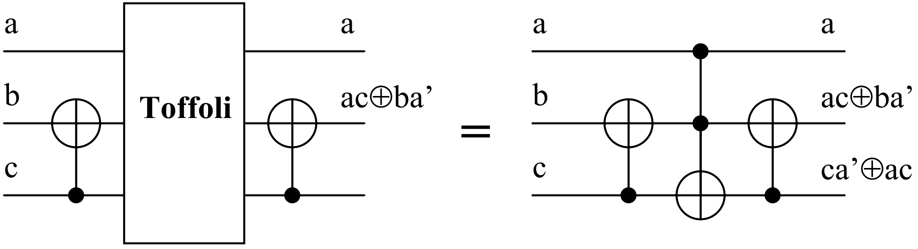


Figure 5: Fredkin gate realization using one Toffoli and two Feynman gates. (a) Toffoli shown schematically, (b) Toffoli with EXOR-down schematics

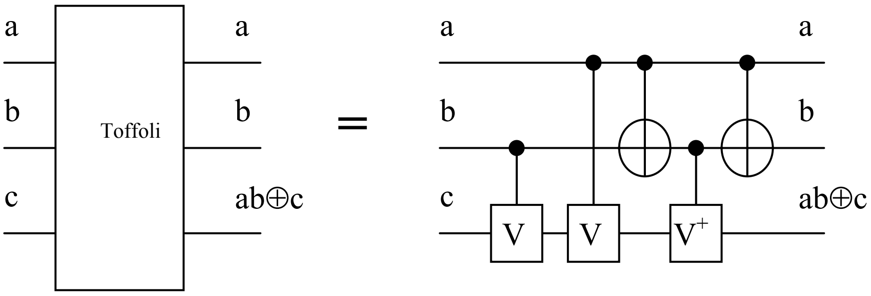


Figure 6: Toffoli gate synthesized using only 2-qubit primitives as proposed by Smolin.

behavior of the entire gate. Observe that Controlled-V and Controlled-V^+ quantum primitives are used. V is the unitary matrix called Square-Root-of-Not, thus a serial connection of gates V realizes an inverter: V*V = NOT. Matrix V^+ is the hermitian of V. As we remember from the definition of hermitian, we have that V * V^+ = I which is the identity matrix (describing a quantum wire). The Controlled-U gate (Figure 7) works as follows: when the controlling signal ***a*** (upper signal in gate Controlled-U) is 0, the output Q of the lower gate repeats its input ***b***. When the controlling signal ***a*** is 1, the output of the lower gate is the operator of this gate (matrix U) applied to its input ***b*** (Figure 7). Therefore, in case of the Controlled-V gate, the output is operator V applied to the input. In case of Controlled-V^+ gate, the output is operator V^+ applied to the input of the gate. The reader can verify in Figure 6b that when ***a***=1, ***b***=0, the operation V * V^+ = I is executed in line ***c***, thus the output of this gate repeats the value of ***c***. When ***a*** = 1 and ***b*** = 1, the operation V * V =NOT is executed in line c, thus the lowest output is a negation of input ***c***. Which is the same as ***ab*** ⊕ ***c*** executed in on the corresponding qubit in Toffoli (Figure 6a). Other output values are also the same as in the Toffoli gate equations. The reader is asked to analyze the operation of this gate on the remaining input combinations to verify its correctness. Let us also remember that Feynman gate has two variants. The EXOR gate can be on an upper or on a lower wire. Similarly, there are three 3-qubit Toffoli gates, EXOR-up, EXOR-down and EXOR-middle (Figure 3). Observe that these gates have all different unitary matrices so they are formally different quantum gates.

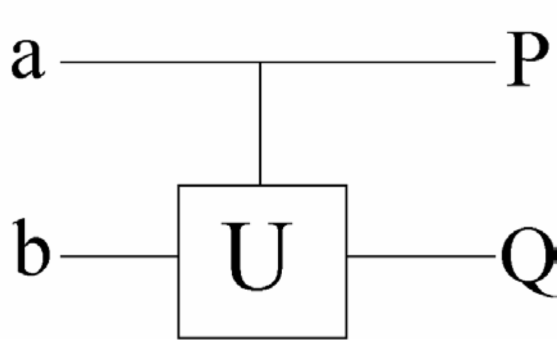


Figure 7. A general-purpose Controlled Gate. When a = 0 then Q = b, when a = 1, then Q = U(b).

3. Evolutionary Approaches to Synthesis of Quantum Arrays

There are basically two methods of designing and drawing quantum circuits. In the first method you draw a circuit from gates and you connect these gates by standard wires. This is like in classical circuit design, but **your gates are reversible or quantum gates**. The rules to design a reversible circuit using this approach are the following: (1) no loops allowed in the circuit and no loops internal to gates, (2) fan-out of every gate is one (which means, every output of a gate can be connected only to one input terminal). These rules preserve the reversible characteristic of gates thus the resulting circuit is also completely reversible. Next, when you draw the circuit, the gates are placed on a 2-dimensional space and the connections between them are routed. Every crossing of two wires in the schematics is replaced with the quantum Swap gate making the schematics planar, which means, no more two wires intersect in it. The schematics is thus rewritten to a quantum array notation illustrated below. It is relatively easy to transform a quantum array to its corresponding unitary matrix, as will be illustrated in the sequel. Kronecker and standard matrix multiplications are used in this transformation. A unitary matrix of a parallel connection of gates A and B is the Kronecker product A⊗B of matrices A and B. A unitary matrix of a serial connection of gates A and B is the standard matrix product A * B of matrices A and B. Transformation of quantum array to its unitary matrix is done for analysis or verification and is a part of our evolutionary algorithm. The approaches that use this first design method of reversible circuits are closer to those of the classical digital CAD where the phases of logic synthesis and physical (geometrical) design are separated.

The second design method for quantum circuits is to synthesize directly the quantum array of a circuit that was initially specified by a unitary matrix. This is done without involving additional graph-based or equation-based representations. The synthesis is done by one of two approaches:

(a) composing matrices of elementary gates in series or in parallel until the matrix of the entire circuit becomes the same as the specification matrix,

(b) decomposing the specification matrix of the entire circuit to parallel and serial connections of unitary matrices until all matrices correspond to matrices of elementary gates directly realizable in the given technology. For simplification, from now on we will talk interchangeably about a gate and its unitary matrix.

In another synthesis variant, called the approximate synthesis, it is not required that the circuit specification matrix and the matrix of composed gates are exactly the same. They can differ by small allowed values or/and differ in some matrix coordinates only.

The synthesis problem of a quantum array has therefore a simple formulation as a composition or decomposition problem, but practical realization of synthesis algorithms for quantum circuits is extremely hard. Observe that even in the case of standard (irreversible) binary logic these composition and decomposition approaches are computationally very difficult and it is only very recently that they are becoming practical in CAD software, because of their high computational complexity. In the case of quantum logic the situation is much more difficult, because of the fast increase of data sizes and because so far the mathematics of description transformations is limited, heuristics are not known and there are no counterparts in quantum logic of such familiar notions as Karnaugh Maps, prime implicants or reductions to covering/coloring combinatorial approaches. Therefore most authors turned to evolutionary algorithms as the fast prototyping methods for quantum arrays [1,2,3,5,6,8,54]. These approaches seem to be good for introductory investigations of the solution space and its properties, with the hope that by analyzing solutions we will learn more about the search space and ultimately create more efficient algorithms. As we will see below, this phenomenon actually happened in case of our research.

In the evolutionary computation approach to quantum synthesis, two key elements strongly influence the convergence of the search. These are: (1) the evaluation function (called also the fitness function), and (2) the encoding of the individuals [9]. Genetic programming uses the encoding of data to trees so the operations on them are very time-consuming [36]. Here we propose a new way to calculate costs of gates and circuits as a part of the fitness function. Also proposed is a new method of QC and RL circuit encoding in a GA. Since we use the same encoding for QC and RL and for many variants of algorithms, design parameters and cost functions, we can easily compare respective synthesis results under strictly the same conditions. This has been not accomplished by previous authors.

The synthesis of quantum logic circuits using evolutionary approaches has two fundamental aspects.

First, it is necessary to find the circuit that either (A) exactly corresponds to the specification, or (B) differs only slightly from the specification. Case (A) is verified by a tautology of the specification function and solution function. In case of a truly quantum circuit this is done by a comparison of unitary matrices. In case of permutation functions this can be also done by comparing the truth tables. Observe that non-permutation matrices cannot be represented by truth tables which leaves the representation of unitary matrices as the only canonical function representation. This representation is responsible for less efficient tautology verification during fitness function calculations, which considerably slows down the software. Case (B) for permutation circuits is verified by an incomplete tautology (tautology with accuracy to all combinations of input values and with arbitrary logic values for don't care combinations). In some applications such as robot control or Machine Learning it is sufficient that the specification and the solution are close, like, for instance, differing only in few input value combinations.

Second fundamental aspect of quantum logic synthesis is that the size (cost) of the circuit has to be minimal, in order to allow the least expensive possible quantum hardware implementation (like the minimum number of electromagnetic pulses in NMR technology). The cost differs for various technologies, but some approximate costs functions will be formulated here that are still more accurate than those previously proposed in the literature.

Consequently, the fitness function of the GA should take into account both the above-mentioned aspects, which means the GA should be designed so as to minimize both the distance from the specification and the final circuit cost simultaneously. Sometimes we are not interested only in those solutions that meet the specification exactly. Usually we are not interested in the solution that would have the exact minimal cost. Multiple designs of a GA are tested here and their results are compared in order to select the minimum cost solution. This approach not only improves the cost of any solution found, but also helps us to develop gradually better generations of our evolutionary/algorithmic/heuristic programming tools for QC CAD.

In our software the fitness function is optimized for the minimal cost, speed of synthesis or precision of synthesis. In case of designing a reversible circuit, the fitness function can be also optimized for the speed of synthesis, the cost of the circuit or for the total number of wires (insertion of a minimum number of constants). These two methods of optimization for each technology and logic type allow to formulate precise constraints for the synthesis method in use.

In this paper we derive more precise cost functions for gates to be used in the optimization algorithm. We will follow the footsteps of some previous papers in quantum computing and we will realize all gates from 1* 1 gates (called also 1-qubit gates) and 2*2 gates (i.e. 2-qubit gates). Moreover, according to [21] we will assume that the cost of every 2*2 gate is the same. Although our methods are general, to be more specific we assume Nuclear Magnetic Resonance (NMR) quantum computers [4,49,50,51,52], for which we approximately assume costs of all 1-qubit gates to be 1, and 2-qubit gates to be 5. Thus, every quantum gate (particularly, every permutation quantum gate) will be build from only 1-qubit and 2-qubit primitives and its cost respectively calculated as the total cost of these primitives used in the circuit. Obviously, this approach can be used for both QC and RC, since in evolutionary approaches we can always arbitrarily restrict the gates used for synthesis, and in case of RC only a subset of permutation gates can be selected.

4. Costs of quantum gates.

An important problem, not discussed so far by other authors, is the selection of the cost function to evaluate the QC (or RC) designs. Although the detailed costs depend on any particular realization technology of quantum logic, so that the cost ratios between for instance Fredkin and Toffoli gates can differ in NMR and ion trap realizations, the assumptions used in several previous papers [2,3,4,5,6,8]; that each gate costs the same, or that the cost of a gate is proportional to the number of inputs/outputs, are both far too approximate. These kinds of cost functions do not provide any basis for the synthesis optimization and as shown by us [1, 2] can lead to quantum sequences for NMR that are far from the minimum. In this section some of the well-known decompositions of known 3-qubit quantum permutation gates, as well as the new gates, will be discussed to illustrate this problem.

The cost of quantum gates in synthesis methods can be seen from as many points of view as there are possible technologies. Previous researches used specific cost to minimize the number of wires per gate, the number of gates in the circuit or other task-specific criteria. Assume a random function to be synthesized in QL and defined over a set of basic variables a, b, c and d. This function will use 4 qubits and the unitary matrix representation of the circuit will be of size $2^4 * 2^4$. The matrix representation can be also called the evolution matrix of the system [10] but this name has nothing to do with evolutionary synthesis algorithms. Defined as quantum evolutionary processes, there is theoretically an infinity of possible quantum gates. In the present work only well-known and widely used unitary quantum gates, as those listed above, will be used. The user of our software can however very easily extend the system by defining arbitrary gates as 256-character symbols together with their unitary matrices and natural numbers for costs. This applies also to multi-qubit gates and multi-valued quantum gates [55].

Based on NMR literature [49,50,51,52] we assume the 1-qubit gates to have a cost of 1 and the 2-qubit gates to have the cost of 5. This leads to 3-qubit gate costs of about 25. From this cost attribution it is evident that the optimization should be aimed toward the use of gates with as few qubits as possible. The cost of the circuit is the total cost of gates used. Of course, especially using GAs, the search might not always be successful but a solution will have always two evaluation criteria: the final circuit error and the final cost calculated after the optimizing "local equivalence" transformations being applied to the initial circuit produced by a GA. (Sometimes we use in GA more precise gate costs if they are known from the literature, but the costs given above will be used for illustration purposes here).

In the ideal case the GA will find an optimal circuit; i.e. the circuit with the smallest possible cost and with the correct matrix representation. Two worse alternatives are: (1) a circuit with a higher cost than the minimum but still with a correct matrix, (2) a circuit with not completely correct matrix but with a smaller cost. For future use let's refer to these three groups of alternatives as C_o (optimal cost), C_a (average cost) and C_w (worst cost), respectively. In the case of RL circuit synthesis the differences in results can occur on one more level: as mentioned before, input constants insertion is allowed in RL synthesis. However, we want to minimize the number of such constants in order to reduce the circuit's width. Once again, as will be shown below, this parameter can be also taken into account by our software as part of the cost function. Consequently, one can obtain a correct circuit with a small cost of gates but with a higher number of wires.

Cost of the result in gates C_g	Function found	Total Cost C_t
1. $\text{Min} \leq C_g$	YES	$\text{Min} \leq C_t$
2. $\text{Min} < C_g < \text{Max}$	YES	$\text{Min} < C_t < \text{Max}$
3. $C_g \leq \text{Max}$	YES	$\text{Min} < C_t < \text{Max}$
4. $\text{Min} \leq C_g$	NO	$\text{Min} < C_t < \text{Max}$
5. $\text{Min} < C_g < \text{Max}$	NO	$\text{Min} < C_t < \text{Max}$
6. $C_g \leq \text{Max}$	NO	$C_t \leq \text{Max}$

Table 1: Illustration of possible results using a combination of cost function per gate C_g and a global evaluation cost function C_t. The column in the middle indicates if the searched function was found or not found.

Table 1 systematizes possible results of a Quantum Circuit synthesis. In the table possible search solutions are characterized with respect to the combinations of the two cost functions.

The cost C_g is the cost of the circuit based on the sum of costs of all gates. C_t is the total cost where the correctness of the result is also included, based on one of the previously explained verification methods between the specification and the obtained circuit descriptions. As will be seen, both of these two cost functions can be handled in order to accelerate or moderate the speed of convergence of the entire synthesis process. A closer look at the table will reveal three categories of results as previously described. These are circuits with small, average and high total costs, respectively. These types can be again separated in two, depending whether the final solution was found or not. Groups 1, 2, and 3 are for exact design, groups 4, 5, and 6 for approximate. Groups 1 and 4 correspond to C_0, Groups 2 and 4 to C_a, and groups 3 and 6 to $C_{w.}$ The category with the most circuits is the one with an average cost, and as will be seen later it is this category that needs particular attention while selecting and adapting a fitness function for a synthesis problem. Min and Max in the table are some parameters set by the user and based on experience.

Now let us continue the discussion of the circuit decomposition into primitives. Let us consider one practical example. The Toffoli or Fredkin gates introduced in section 2 are both universal quantum logic gates that are well-known. They have been built in several quantum and reversible technologies. The problem is to find an optimal decomposition of the universal gates into smaller parts, especially the directly realizable quantum primitives such as Feynman, NOT or Controlled-V (C_V) gates. As mentioned earlier, the gates with one and two qubits have costs 1 and 5, respectively. Consequently the assumption of using only 1-qubit and 2-qubits gates will be observed since only such gates are *directly realizable in quantum hardware.* Other gates are only abstract concepts useful in synthesis and not necessarily the best gates for any particular quantum technology (like NMR in our case). Figure 6 presented the well-known realization of Toffoli gate from [21]. According to Figure 6b there are five 2-qubit primitives, so the cost is 5 * 5 = 25.

Now that we learned in section 2 how Toffoli gate works internally based on the operation of its component quantum primitives, we will realize the Fredkin gate from the Toffoli gate. Using a GA [1,2] or the synthesis method from this paper, we can synthesize the Fredkin gate using two Feynman gates and one Toffoli gate as in Figure 5. The cost of this gate is 2*5 + 25 = 35.

Substituting the Toffoli design from Figure 6b to Figure 5 we obtain the circuit from Figure 8a. Now we can apply an obvious EXOR-based transformation to transform this circuit to the circuit from Figure 8b. This is done by shifting the last gate at right (Feynman with EXOR up) by one gate to left. The reader can verify that this transformation did not change logic functions realized by any of the outputs. Observe that a cascade of two 2*2 gates is another 2*2 gate, so by combining a Feynman with EXOR-up gate (cost of 5), followed by controlled-V gate (cost of 5) we obtain a new gate C_V with the cost of 5. Similarly gate C_V$^+$ with cost 5 is created. This way, a circuit from Figure 8c is obtained with the cost of 25. (This transformation is based on the method from [21] and the details of cost calculation of C_V and C_V$^+$ are not necessary here). Thus, the cost of Toffoli gate is exactly the same as the cost of Fredkin gate, and not half of it, as previously assumed and as may be suggested by classical binary equations of such gates.

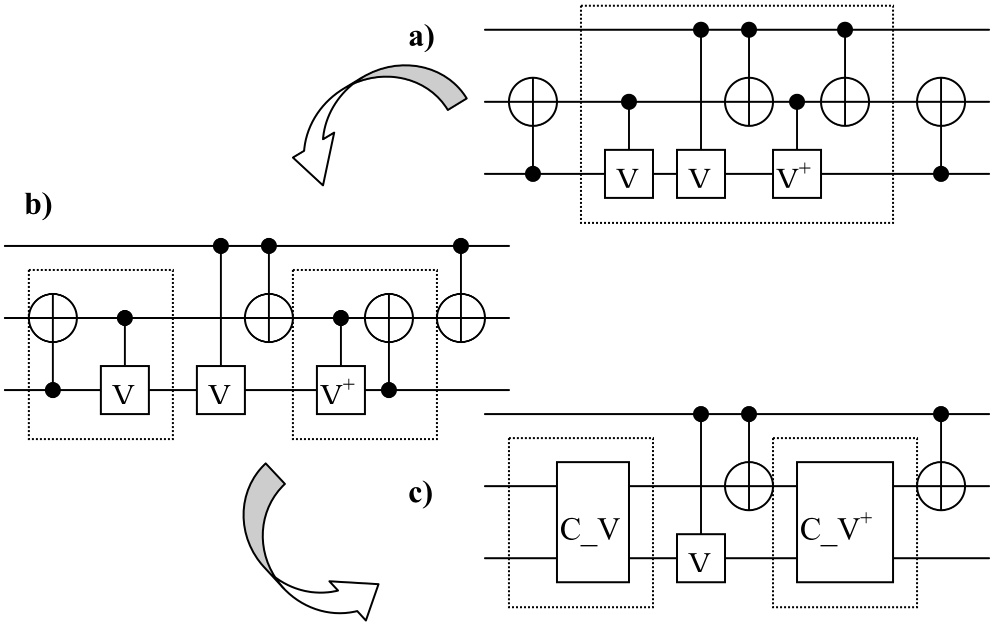


Figure 8: Example of reducing cost in the Fredkin gate realized with quantum primitives. Gates C_V and C_V$^+$ in Figure 8c are created by combining respective quantum primitives from Figure 8b which are shown in dotted lines.

Encouraged with the above observation that sequences of gates on the same quantum wires have the cost of only single gate on these wires, we used the same method to calculate costs of other well-known gates. Let us first investigate a function of three majorities investigated by Miller [22,23,56]. This gate is described by equations: P = ab ⊕ ac ⊕ bc, Q = a’b ⊕ a’c ⊕ bc, P = ab’ ⊕ ac ⊕ b’c. Where a’ is a negation of variable a. Function P is a standard majority and Q, R are majorities on negated input arguments a and b, respectively [56]. We realized this function with quantum primitives, found it useful in other designs and thus worthy to be a stand-alone quantum gate. We call it the Miller gate [56]. As seen in Figure 9a, the Miller gate requires 4 Feynman gates and a Toffoli gate, which would suggest a cost of 4*5 + 25 = 45. However, performing transformations as in Figure 9b, we obtain a solution with cost 35. Another solution obtained by the same method has cost 35 and is shown in Figure 9c. It is also based on simple EXOR transformation $(x \oplus y) \oplus z = (x \oplus z) \oplus y$ applied to three rightmost Feynman gates from Figure 9a, with EXOR in the middle wire y. (More on optimizing equivalence-based transformations in section 6). Again, the Miller gate, based on its binary equations, looks initially much more complicated than the Toffoli gate, but a closer inspection using quantum logic primitives proves that it is just slightly more expensive.

5. Our Entire minimization methodology of quantum arrays synthesis.

Based on examples from section 4, let us observe that a new permutation quantum gate with equations:

$P = a$,
$Q = a \oplus b$,
$R = ab \oplus c$

can be realized with cost 20. It is just like a Toffoli gate from Figure 6b but without the last Feynman gate from the right. This is the cheapest quantum realization known to us of a complete (universal) gate for 3-qubit permutation gate (Figure 10). It is thus worthy further investigations. We found that the equation of this gate was known to Peres [35], but it has not been used in practical designs of quantum or reversible circuits. We propose here to consider the Peres gate as a base of synthesis and transforms, similarly as 2-input NAND gates are used in technology mapping phase of classical Boolean CAD.

Observe that assuming the availability of Peres gate, the algorithms from literature (for instance [25,26]) will not lead to a quantum array with minimum quantum costs formulated as above. When given the unitary matrix of the Peres gate, these algorithms would return the solution of cost 30 composed from one Toffoli gate and one Feynman gate, which would lead to clearly non-minimal electromagnetic pulse sequence. Thus, improved synthesis algorithms should be created to minimize the realistic quantum gate costs introduced in [21] and in this paper. Observe please also, that if a post-processing stage were added to the algorithms from [22,23,24,25,26] (or to the result of the GA from section 7), then the optimum solution (of a single Peres gate and cost of 20) would be found for this example.

Therefore our entire synthesis and optimization approach to quantum arrays is the following.

1. First create the circuit by some approximate synthesis method (evolutionary algorithm in our case),
2. Apply macro-generation of complex gates to quantum primitives,
3. Apply the optimizing equivalence-based transformations.

The application of this procedure to our example will have the following stages:

(a) the initial solution is a Toffoli gate followed by a Feynman gate (with EXOR down) on quantum wires 1,2 (Figure 10b).

(b) the macro-generation of the Toffoli gate leads to a Peres gate followed by the Feynman gate (with EXOR down) on quantum wires 1,2.

(c) the equivalence-based transformation (presented below) will find a pattern of two Feynman gates of the same type (EXOR down) in sequence, which is replaced with two quantum wires, 1 and 2 (Figure 10c).

(d) thus the two right Feynman gates are cancelled, the same way as two inverters in series are cancelled in standard logic synthesis.

(e) the resultant circuit, a final solution, will have just one Peres gate of cost 20 (Figure 10d).

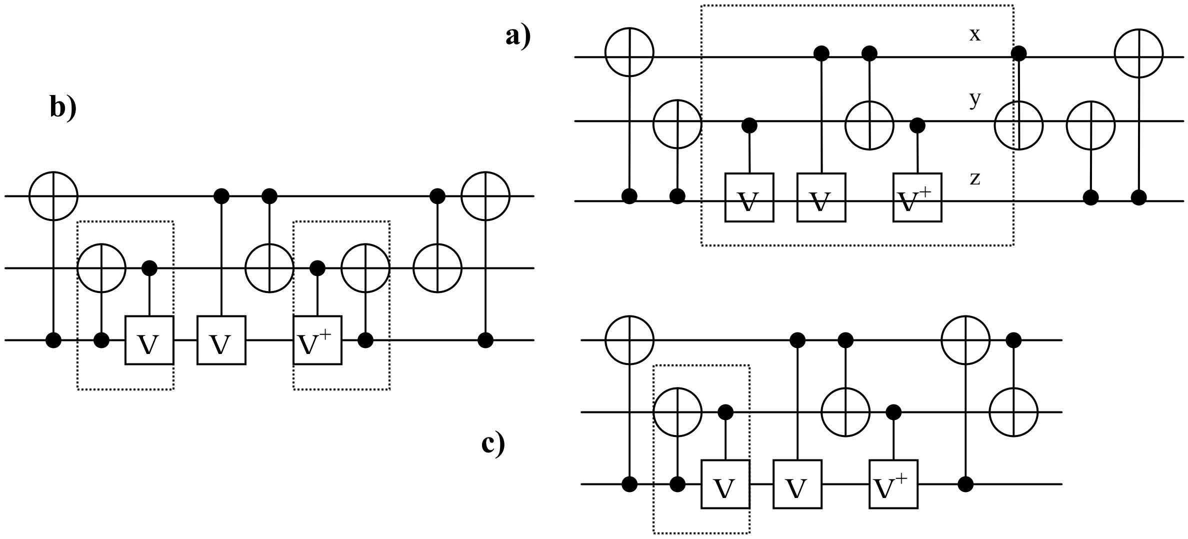


Figure 9: Reducing the cost of realization of Miller gate using quantum primitives: Feynman EXOR up, Feynman EXOR down, Controlled-V (V down), and Controlled V^+ (V^+ down)

. As can be seen from previous examples, the gates matrices have infinity of possible representations in QL. The above samples of synthesis are situated in the C_o category of Table 1, because if eventually a smaller circuit of a Fredkin gate were found, then the one presented here will remain very close to the minimal cost. Consequently the group that needs the closest inspection is the C_a because it includes circuits of great interest. As the number of circuits is infinite, the circuits in the last group can be considered as being too far from the searched solution. However the circuits in the group having average cost (not too far from the solution and not too big) can be simplified to better ones, and also they can already include parts of the optimal circuits.

6. Local Transformations.

The transformations are grouped in 12 transformation sets. There are the following sets:

S1. 1-qubit transformations,

S2. 2-qubit transformations,
S3. 3-qubit transformations,
S4. 4-qubit transformations,
S5. n-qubit transformations,
S6. Ternary transformations,
S7. Mixed binary/ternary transformations,
S8. Macro-generations,
S9. Macro-cell creations,
S10. Peres Base transformations,
S11. Toffoli Base transformations,
S12. Controlled-V Base transformations,
S13. Input/Output permutting transformations.

Many transformations are shared between sets. In addition, in each of the above base sets, there are subsets to be chosen for any particular run of the optimizer program. Most of them are taken from [22-26,49,53] but some other are based on our research or general quantum literature. Different groups of transformations are used in various stages of circuit optimization. The 1-qubit transformations are related to 1-qubit gates (Tables 2,3 and 4). They can precede and also follow the 2-qubit, 3-qubit and other transformations. The 2-qubit transformations are for 2-qubit circuits or 2-qubit subcircuits of larger quantum circuits, similarly the 3-qubit transformations are for 3-qubit circuits or subcircuits of larger circuits (Table 5). The n-qubit transformations are general transformation patterns applicable to circuits with more than 3 qubits. They are less computationally efficient and they use internally transforms S1 – S4. Ternary transformations (only 1-qubit, 2-qubit and 3-qubit) are used for ternary quantum logic synthesis and mixed transformations for mixed binary/ternary quantum

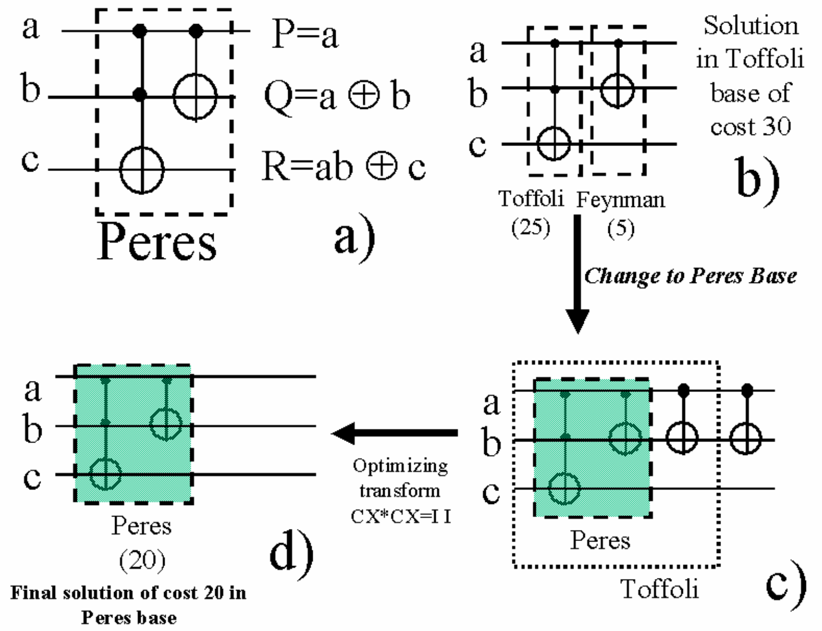


Figure 10. Peres gate: (a) Peres gate as a block using Toffoli Base notation, (b) Solution to Peres gate in Toffoli Base with cost 30, (c) Result of macro-generation of circuit from Figure 10b to Peres base, (d) After applying the optimizing transform that removes a successive pair of Feynman gates. The cost is now 20.

logic synthesis [57]. Macro-generations are transformations that convert higher order gates such as Fredkin, Margolus, DeVos, Kerntopf or Perkowski gates to standard bases. Macro-cell creations from set S9 are inverse to those from set S8. There are three standard bases of transformations: Toffoli Base, Peres Base, and Controlled-V Base. In Toffoli Base all permutation gates are converted to X (i.e. NOT), 3-qubit Toffoli and 2-qubit Feynman gates. This is the standard synthesis base used by all other authors in literature [22-26,48]. The Peres Base has been introduced originally by us based on the observation of superiority of this base in NMR realizations (and perhaps other realizations as well). It includes only X, 3-qubit Peres and 2-qubit Feynman gates. Controlled-V Base is another new base that is very useful to synthesize new low-cost permutation gates from truly quantum primitives of limited type [55,57]. This base includes Controlled-V, Controlled-V^+, V, V^+, X and Feynman gates. In all bases the gates like Feynman, Toffoli, Controlled-V are stored in all possible permutations of quantum wires. Thus in 2-qubit base there are "Feynman EXOR up" and "Feynman EXOR down" gates and transformations respective to each of these gates. For simplification, in the tables below only some of the transformations are shown, for instance related only to "Feynman EXOR down" or "Toffoli EXOR down gates". Other transformations are completely analogous. The output permutting transformations lead in principle to a circuit that has an unitary matrix which is different from the original unitary matrix. Observe that each transformation can be applied forward or backward, so the software should have some mechanisms to avoid infinite loops of transformations. The matrix of the new circuit is the matrix of the original circuit with permuted output signals. In some applications the order of output functions is not important, so if the circuit is simplified by changing the output order, the output permutting transformation is applied.

In addition to operators defined earlier, we define now the following operators:

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{j\pi}{4}} \end{bmatrix}, \quad P(\phi) = e^{\frac{j\phi}{2}} I, \quad X(\phi) = \cos\frac{\phi}{2} I - j \sin\frac{\phi}{2} X,$$

$$Y(\phi) = \cos\frac{\phi}{2} I - j \sin\frac{\phi}{2} Y, \quad Z(\phi) = \cos\frac{\phi}{2} I - j \sin\frac{\phi}{2} Z$$

X,Y,Z defined earlier are Pauli spin matrices and $X(\phi)$,$Y(\phi)$, $Z(\phi)$ are the corresponding exponential matrices, giving rotations on the Bloch sphere [48]. P is a phase rotation by $\phi/2$ to help match identities automatically [53].

The transformation software operates on sequences of symbols that represent gates and their parts. Symbol * is used to create sequences from subsequences. This symbol thus separates two serially connected gates or blocks. Numerically, it corresponds to standard matrix multiplication. Gate symbols within a parallel block may be separated by spaces, but it is necessary only if lack of space will lead to a confusion, otherwise space symbol can be omitted. So for instance symbols of macro-cells should be separated by spaces. There are four types of symbols: simple, rotational, parameterized and controlled. Simple symbols are just names (here – single characters, in software – character strings). The names of simple symbols (used also in other types of symbols) are the following: D – standard control point (a black dot in an array), E – control with negated input, F – control with negated output, G – control

with negated input and output, X – Pauli-X, Y – Pauli-Y, Z – Pauli-Z, H – Hadamard, S – phase, T – Π/8 (although Π/4 appears in it). Symbols A, B and C are auxiliary symbols that can match several gate symbol definitions. They do not correspond to any particular gates but to groups of gates and are useful to decrease the set of rules and thus speed-up transformations. Other simple symbols will be explained below. As we see, characters are used here not only for gates but also for parts of gates, such as D – standard control used in gates. Thus we can combine these symbols to create gate descriptions: DX (Feynman with EXOR down), XD (Feynman with EXOR up), DDX (Toffoli or Toffoli with EXOR down), DXD (Toffoli with EXOR in middle), XDD (Toffoli with EXOR up), and so on. Names of macro-cells are for instance: FE (Feyman), TOD (Toffoli with EXOR down), SW (swap), FRU (Fredkin controlled with upper wire), MA (Margolus), KED (Kerntopf with Shannon expansion in lowest wire), etc. Symbols like ϕ, θ, π denote angles and other parameters. Parameterized symbols have the syntax:
simple name ***[parameter$_1$,…parameter$_n$],*** where ***parameter$_i$*** are parameters. For instance, X[4Π/8], Y[3Π/2], Z[ϕ], etc. Rotational symbols are composed of the (simple) rotation operator symbol such as X, Y, Z, or P, and a number. X[ϕ_i], Y[ϕ_i], Z[ϕ_i], P[ϕ_i], where ϕ_i = 4Π/8 * i , i = 1,2,…,7. We assume here that all rotational operators have period 4Π and equal identity when their argument is 0, thus the choices for ϕ_I . In these operators the notation is like this, Xr = X[4Π/8 * r], r = 1,2,…,7, and so on for Yr, Zr and Pr. Controlled symbols have the syntax ***simple_name*** **[** ***sequence of simple,rotational or parameterized symbols*** **]**
Example 1: D[X*X] is a symbol of a controlled gate that is created from two subsequent Feynman gates. Observe that DX*CX = D[X*X] = D[I] = I I, which means that two controlled-NOT gates in sequence are replaced by two parallel quantum wires denoted by I I. Example 2. D[S*T*H] is a sequence S*T*H controlled by single qubit. Example 3. DD[X1*Y2*Y2] is a sequence of rotational gates controlled by a logic AND of two qubits (this is a generalization of a Toffoli gate). Observe that controlled symbols are created only as a transitional step during the optimization process – such gates do not physically exist. Using the concept of controlled symbols, n-qubit circuits can be optimized using 1-qubit transformations without duplicating all the 1-qubit identity rules. Similarly the parameterized and rotational symbols allow the reduction and hierarchization of the set of rules, which causes more efficient and effective run of the optimization software.

The simplified algorithm SA for performing rule-based optimization of 3-qubit quantum arrays is the following:

1. Apply all the 1-qubit transformations, until no more applications of such rules becomes possible.
2. Apply all the 2-qubit transformations and 1-qubit transformations induced by them (for instance using the controlled symbols).
3. Apply all the 3-qubit transformations until possible.
4. Iterate steps 1,2 and 3 until no changes in the circuit.
5. Apply inverse transformations that locally optimize the array.
6. Repeat steps 1,2,3,4 until possible.
7. Apply inverse transformations that do not worsen the cost of the array.
8. Repeat steps 1,2,3,4 until possible.

Several similar variants of this heuristic algorithm can be created. In general, none of these versions gives a warranty of the optimal or even sub-optimal solution, as known from the theory of Post/Markov algorithms.

As an example, we present 1-qubit transformation algorithm A1q:

A1. Combine modulo-8 the same types of rotational operators P,X,Y,Z.
For instance X2*X3 becomes X5, X2*X3*X3 becomes I,
and Y3*Y3*Y3 becomes I*Y1=Y1.

A2. Apply directly applicable rules that do not include symbols A and B.

A3. Iterate 1 and 2 until no more changes possible.

A4. Starting from the left of the sequence, find a grouping pattern such as A2 = - BC

A5. Substitute symbols X,Y,Z for A,B and C from the pattern.

A6. Apply in forward directions the standard simplifying transformations such as I*A=A

A7. Repeat steps A1 to A6 until possible.

Example 4. [53]. Given is an identity Y = - X*Y*X. Let us try to verify this identity using algorithm A1q. We have therefore to simplify the sequence X*Y*X. (A1) There are no patterns of the same rotational operators to combine, (A2) There are no directly applicable rules, (A4) We take pattern -X*Y and match it with the rule A2 = C*B. This leads to – Z2*X. (A1) no, (A2) no, (A3) no, (A4) we find pattern A2 = CB which leads to –Y*X*X. (A6) X*X is replaced with I, Y*I is replaced with Y. No further optimization steps are possible, so the sequence was simplified to –Y, proving that Y = -Y*Y*X.

Example 5. Simplify HXH. (A2) Use rule H = X*Y1 twice. This leads to X*Y1*X*X*Y1. (A2) Use rule I=AA in reverse direction. This leads to X*Y1*Y1. (A1) Y1*Y1 is replaced by Y2. This leads to X*Y2. (A4) Use pattern A=B*C2. This leads to Z. No further optimization steps are possible. Thus we proved that HXH = Z.
More optimization examples of algorithm SA will be given in the sequel.
In our runs of GA we look for solutions with the accuracy of: (1) permutation of quantum wires, (2) permutation of inputs, (3) permutation of outputs (transformation group S13). In addition we can also generate solution sets with accuracy of inverting input, output or input/output signals (the NPN classification equivalent circuits). Therefore, for each unitary matrix we generate therefore many logically equivalent solutions. We can generate solution sets also for the same set of Boolean functions, or for the same NPN classification class. One interesting aspect of such approach is that one can create new local equivalence transformations for circuits in each of these classes. Finding these transformations and applying them exhaustively to particularly interesting gates leads to levellized “onion-like” structures of gates, as the one shown in Figure 11. This Figure shows the layered structure of gates created by adding only Feynman gates to a seed composed of other gate types, in this case a Toffoli gate. Figure 12 shows the layered structure with Peres gate as a seed, in which Toffoli, Fredkin and Miller gates are created. These transformations are used to find efficient realizations of new gates from known gate realizations [55,57].

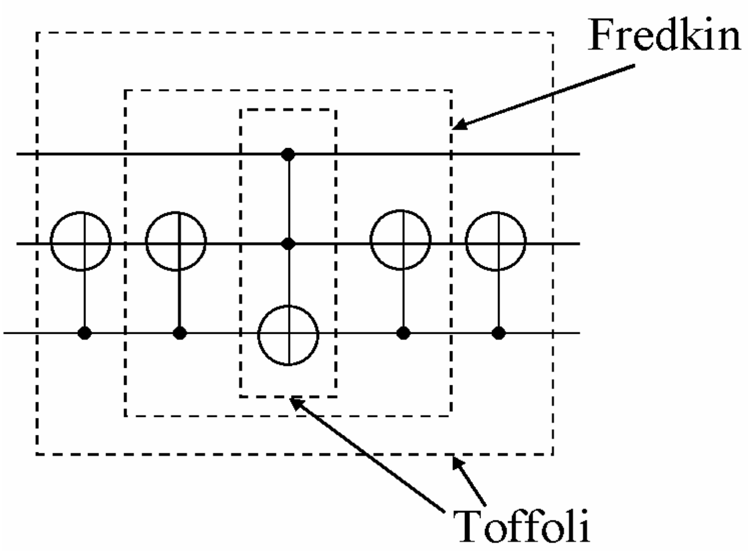


Figure 11. The layered structure of gates. Fredkin from Toffoli and Toffoli from Fredkin.

I1.	I = A*A
I2.	I = -A*C2*B
I3.	I = HH
I4.	I = - H*Y3*X
I5.	I = P2*A2*A
I6.	I = P2*H*X2*Y1
I7.	I = - P2*H*Y3*X2
I8.	I = - P2*H*Z2*Y3
I9.	I = P2*Y1*Z2*H
I10.	I = P2*Y2*Y
I11.	I = - P2*Y3*H*Z2
I12.	I = - P2*Y3*X2*H
I13.	I = P2*Z2*Z
I14.	I = P3*Z3*S
I15.	I = P4*A4
I16.	I = - Y*X2*Z
I17.	I = - Z*Y2*X
I18.	I = - Z*Y3*H

A1.	A = B*C2
A2.	A = B2*C
A3.	A = B3*A*B3
A4.	A = B3*C*B1
A5.	A = - C*B2
A6.	A = C1*B*C3
A7.	A = - C2*B
A8.	A = C3*A*C3
A9.	A = P2*A2
A1.1.	A1 = - B*A1*C
A1.2.	A1 = - B*A3*B
A1.3.	A1 = - C*A3*C
A1.4.	A1 = P2*A3*A
A2.1.	A2 = - B*C
A2.2.	A2 = CB
A2.3.	A2 = - P2*A
A3.1.	A3 = B*A3*C
A3.2.	A3 = - C*A3*B
A3.3.	A3 = - P2*A1*A
A4.1.	A4 = P4

X1.	X = - H*Y3
X2.	X = S*X2*S
X3.	X = Y1*H
X4.	X = Y3*H*Y2
X5.	X = Z1*X*Z1
X6.	X = - Z3*Y*Z1
X1.1.	X1 = H*Z1*H
X1.2.	X1 = Z*X1*Y
X2.1.	X2 = H*Z2*H
X2.2.	X2 = P2*H*Y3
X2.3.	X2 = - S*X*S
X2.4.	X2 = - Y3*H*Y
X2.5.	X2 = Y3*Y*H
X3.1.	X3 = H*Z3*H
X3.2.	X3 = - S*H*S
X3.3.	X3 = - Y*X1*Y
X3.4.	X3 = - Z*X1*Z

Table 2: 1-qubit transformations for I, A and X groups.

Y1.	Y = - H*X2*Y3
Y2.	Y = H*Y3*Z2
Y3.	Y = - H*Z2*Y1
Y4.	Y = P2*Y2
Y5.	Y = S*Y2*S
Y6.	Y = X1*Y*X1
Y7.	Y = X2*Y3*H
Y8.	Y = X3*Y*X3
Y9.	Y = - X3*Z*X1
Y10.	Y = -Y1*X2*H
Y11.	Y = -Y3*H*X2
Y12.	Y = - Y3*Z2*H
Y13.	Y = - Z1*X*Z3
Y13.	Y = Z1*Y*Z1
Y14.	Y = - Z2*H*Y3
Y15.	Y = Z3*Y*Z3
Y1.1.	Y1 = - H*Y*X2
Y1.2.	Y1 = - H*Y2*X
Y1.3.	Y1 = - H*Y3*H
Y1.4.	Y1 = H*Z
Y1.5.	Y1 = P2*H*Z2
Y1.6.	Y1 = P2*Y3*Y
Y1.7.	Y1 = X*H
Y1.8.	Y1 = Y*H*X2
Y1.9.	Y1 = - Z*Y2*H
Y1.10.	Y1 = Z2*H*Y
Y1.11.	Y1 = - Z2*Y*H

Y2.1.	Y2 = - H*Y2*H
Y2.2.	Y2 = - H*Y3*Z
Y2.3.	Y2 = - P2*Y
Y2.4.	Y2 = - S*Y*S
Y2.5.	Y2 = - X*Y3*H
Y3.1.	Y3 = H*Z2*Y
Y3.2.	Y3 = - H*X
Y3.3.	Y3 = - P2*H*X2
Y3.4.	Y3 = - P2*Y1*Y
Y3.5.	Y3 = X2*H*Y
Y3.6.	Y3 = Y*H*Z2
Y3.7.	Y3 = Y*X2*H
Y3.8.	Y3 = - ZH
Z1.	Z = H*Y1
Z2.	Z = P2*Z2
Z3.	Z = S*S
Z4.	Z = - X1*Y*X3
Z5.	Z = X1*Z*X1
Z6.	Z = X3*Z*X3
Z7.	Z = Y2*H*Y3
Z8.	Z = - Y3*H

Z1.1.	Z1 = H*X1*H
Z1.2.	Z1 = P2*Z3*Z
Z1.3.	Z1 = - P3*S
Z1.4.	Z1 = - X*Z3*X
Z1.5.	Z1 = Y*Z1*X
Z1.6.	Z1 = - Y*Z3*Y
Z2.1.	Z2 = H*X2*H
Z2.2.	Z2 = H*Y3*Y
Z2.3.	Z2 = P2*Y3*H
Z2.4.	Z2 = - P2*Z
Z2.5.	Z2 = - P3*Z1*S
Z2.6.	Z2 = - Y*H*Y3
Z2.7.	Z2 = Y1*Y*H
Z3.1.	Z3 = H*X3*H
Z3.2.	Z3 = - P1*Z*S
Z3.3.	Z3 = - P2*Z1*Z
Z3.4.	Z3 = - P3*Z2*S
Z3.5.	Z3 = - X*Z1*X
Z3.6.	Z3 = X*Z3*Y
Z3.7.	Z3 = - Y*Z1*Y

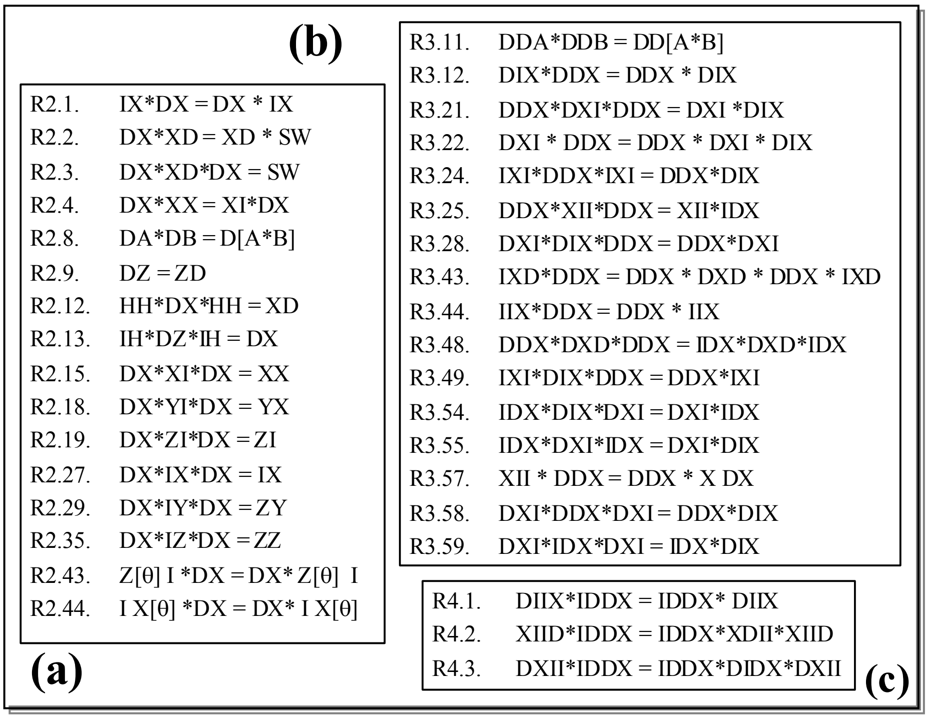
Table 3: 1-qubit transformations for Y and Z groups.

S1.	S = P1*Z1
S2.	S = P3*Z3*Z
S3.	S = T*T
S4.	S = X1*S*Y1
S5.	S = X2*S*Y2
S6.	S = X3*S*Y3
S7.	S = Y*S*X
S8.	S = - Y1*S*X3
S9.	S = - Y2*S*X2
S10.	S = - Y3*S*X1

H1.	H = P2*Y1*Z2
H2.	H = - P2*Y3*X2
H3.	H = S*X1*S
H4.	H = X*Y1
H5.	H = - X1*H*Z3
H6.	H = - X2*H*Z2
H7.	H = - X2*Y3*Y
H8.	H = - X3*H*Z1
H9.	H = Y*X2*Y3
H10.	H = - Y1*Y*X2
H11.	H = - Y1*Z
H12.	H = Y2*H*Y2
H13.	H = Y2*X*Y3
H14.	H = Y3*H*Y3
H15.	H = -Y3*X
H16.	H = Y3*Z*Y2
H17.	H = Y3*Z2*Y
H18.	H = - Z*Y3
H19.	H = - Z1*H*X3
H20.	H = - Z2*H*X2
H21.	H = - Z2*Y1*Y
H22.	H = - Z3*H*X1

X[Π] *Y[ϕ] = Y[- ϕ]*X[Π]
X[-Π]*Y[ϕ] = Y[- ϕ]*X[-Π]
X[ϕ] *Y[Π] = Y[Π] *X[-ϕ]
X[ϕ]*Y[-Π] = Y[-Π]*X[- ϕ]
X[Π/2]*Y[ϕ] = Z[ϕ] *X[Π/2]
X[-Π/2]*Y[ϕ] = Z[-ϕ]*X[-Π/2]
X[ϕ] *Y[Π/2] = Y[Π/2]*Z[ϕ]
X[ϕ] *Y[-Π/2] = Y[-Π/2]*Z[-ϕ]
X[3Π/2]*Y[ϕ] = Z[-ϕ] *X[3Π/2]
X[-3Π/2]*Y[ϕ] = Z[ϕ] *X[-3Π/2]
X[ϕ] *Y[3Π/2] = Y[3Π/2]*Z[-ϕ]
X[ϕ] *Y[-3Π/2] = Y[-3Π/2]*Z[ϕ]
Y[Π] *Z[ϕ] = Z[- ϕ]*Y[Π]

Table 4: 1-qubit transformations for S, H and parameterized rotation groups.



(a)

R2.1.	IX*DX = DX * IX
R2.2.	DX*XD = XD * SW
R2.3.	DX*XD*DX = SW
R2.4.	DX*XX = XI*DX
R2.8.	DA*DB = D[A*B]
R2.9.	DZ = ZD
R2.12.	HH*DX*HH = XD
R2.13.	IH*DZ*IH = DX
R2.15.	DX*XI*DX = XX
R2.18.	DX*YI*DX = YX
R2.19.	DX*ZI*DX = ZI
R2.27.	DX*IX*DX = IX
R2.29.	DX*IY*DX = ZY
R2.35.	DX*IZ*DX = ZZ
R2.43.	Z[θ] I *DX = DX* Z[θ] I
R2.44.	I X[θ] *DX = DX* I X[θ]

(b)

R3.11.	DDA*DDB = DD[A*B]
R3.12.	DIX*DDX = DDX * DIX
R3.21.	DDX*DXI*DDX = DXI *DIX
R3.22.	DXI * DDX = DDX * DXI * DIX
R3.24.	IXI*DDX*IXI = DDX*DIX
R3.25.	DDX*XII*DDX = XII*IDX
R3.28.	DXI*DIX*DDX = DDX*DXI
R3.43.	IXD*DDX = DDX * DXD * DDX * IXD
R3.44.	IIX*DDX = DDX * IIX
R3.48.	DDX*DXD*DDX = IDX*DXD*IDX
R3.49.	IXI*DIX*DDX = DDX*IXI
R3.54.	IDX*DIX*DXI = DXI*IDX
R3.55.	IDX*DXI*IDX = DXI*DIX
R3.57.	XII * DDX = DDX * X DX
R3.58.	DXI*DDX*DXI = DDX*DIX
R3.59.	DXI*IDX*DXI = IDX*DIX

(c)

R4.1.	DIIX*IDDX = IDDX* DIIX
R4.2.	XIID*IDDX = IDDX*XDII*XIID
R4.3.	DXII*IDDX = IDDX*DIDX*DXII

Table 5: Examples of 2-qubit and 3-qubit transformations: (a) 2-qubit transformations, (b) 3-qubit transformations; observe a space between X and DX in rule R3.57 that signifies that D control the lower X, (c) 4-qubit transformations

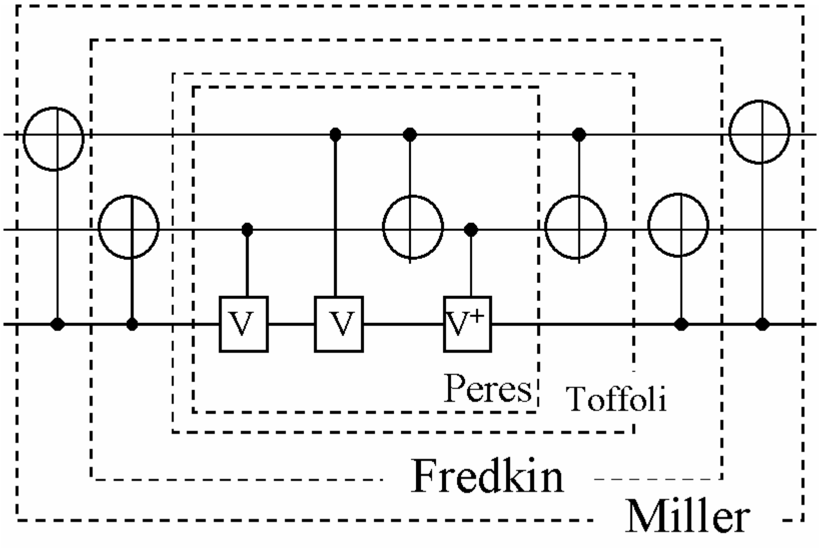


Figure 12: Internal part of a layered structure with Peres gate as a seed.

7. Genetic Algorithm for Quantum Logic Synthesis

Now that we understand all the basic prerequisites for our approach to quantum array synthesis and we have a general plan for it, let us finally discuss the Genetic Algorithm itself. Similarities with genetic information and its transmission in Nature are applied and constitute the core of the evolutionary approach. The application of GA is well known in problems with high noise because their convergence can be moderated according to the fitness function. GAs are widely used optimization algorithms, however only few authors present their application programs for QC synthesis [1,2,3,6].

General steps of using our GA variant for QC (and RL) are the following:

1. Initiate a random population of individuals (here each individual is a Quantum or Reversible circuit). An individual is an encoding of a chromosome of a circuit for fitness function evaluation. The size of the population is generally between 50 and 500 individuals. A too small population will yield a too fast convergence while a too big population will uselessly search a too large problem space.
2. Steps describing the search using the GA are called generations. The search stops when a condition is attained, generally this is a result of having fitness greater than a limit, or that certain number of generations has been attained.
3. Pick n pseudo-randomly selected individuals. The rules of selecting individuals can speed up the convergence of the search, however if the selection operator is too "greedy" (selecting only the best individuals) the algorithm can get stuck in a local minimum. We experimented with several rules, some presented below. They are based on fitness function.
4. For all the individuals of the new population apply two genetic operators:
   a. Crossover – operator allows to recombine two individuals by exchanging parts of their respective chromosomes. This operation is the main power of the search with GAs.
   b. Mutation – operator introducing noise into the GA in order to avoid local minima. Generally applied with a very small probability [0.01, 0.1] (compared to the one of Crossover [0.3, 0.85]) modify randomly a chromosome of a randomly selected individual.
5. Replace the old generation of individuals by the new one.
6. If solution is found then exit, else go to step 2.

All individuals in the GA are quantum circuits partitioned into parallel blocks. Our representation allows describing any Quantum or Reversible circuit [1,2]. This partitioning of the circuit was in our case induced from the representation of any QC such as one in Figure 14. Each block has as many inputs and outputs as the quantum array width (for instance three in Figures 3 – 6). The chromosome of each individual is a string of characters with two types of tags. To indicate to the algorithm where a parallel block begins and ends, a special tag different than any gate was required. This special tag is named here R-tag or 'r'. Gates' tags are characters from the alphabet and in our examples tags are the first letter(s) of gates' names. An example of a chromosome can be seen on Figure 13. The circuit corresponding to the parent 1 from Figure 13 can be seen on Figure 14. In this particular encoding each space (empty wire or a gate) is represented by a tag. Our problem-specific encoding was applied to allow the definition of as simple genetic operators as possible. The advantage of these strings is that they allow encoding an arbitrary QL or RL circuit without any additional parameters known from previous research [5,6]. And only the possibility to move gates, remove and add them to the chromosome consequently make it possible to construct and arbitrarily modify a circuit.

To speed-up the search the crossover operator has only one restriction. Both crossover candidates are restricted to have the same number of wires. This considerably simplifies this operation because it allows to avoid the case when the two circuits are cut in incompatible locations. Consequently, for any circuit with a number of wires higher than one, the crossover is done only between parallel blocks. This assures that all children are well formed and eliminates also the necessity of a repair algorithm. In the case of individuals having only one wire in the circuit, the crossover is executed at a random location in the chromosome.

The mutation is more delicate. Its results should be: adding a gate, replacing a gate by another one, or a removal of a gate. All these operations can be easily implemented except for the replacement operator. Assume the Hadamard gate from circuit in Figure 14 is to be changed to a Feynman gate. It is a particular gate where we need to create a new parallel block because we want to preserve the remaining of the current circuit and also not to create a block with more wires than the circuit actually has. In the remaining cases: (1) the removing of a gate can easily be seen as replacing it by wires, (2) the adding of a gate implies a creation of a new parallel block. The results of genetic operators are shown in Figure 13. In the middle are two parents which yield two children on the left side as a result of a crossover. On the right side are the results of mutation (removal of a gate, adding a new gate and changing of a gate, respectively) on parent 2. As can be seen, the operation of adding a gate results in a creation of a new parallel block, but as will be shown later this operation is not equivalent to the operation of adding one block.

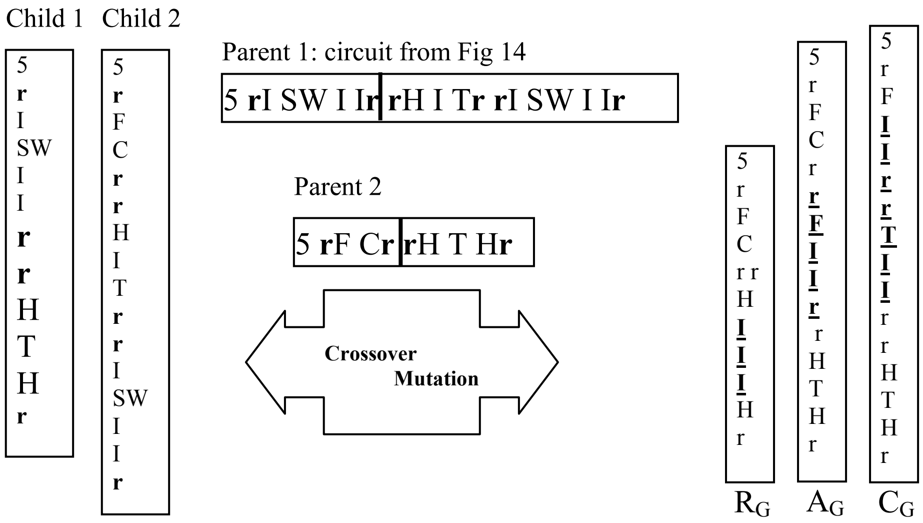

Child 1
Child 2
5 r I SW I I r r H T H r
5 r F C r r H I T r r I SW I I r
Parent 1: circuit from Fig 14
5 rI SW I Ir rH I Tr rI SW I Ir
Parent 2
5 rF Cr rH T Hr
Crossover
Mutation
5 r F C r r H I I I H r
5 r F C r r F I I r r H T H r
5 r F I I r r T I I r r H T H r
RG
AG
CG

Figure 13: Chromosome representing the circuit on figure 14 and genetic operations as example. On the left is a result of crossing-over and on the right is are results of mutation Removal of a gate (R_G), Adding of a gate (A_G) and Replacing of a gate (C_G).

The evaluation of a QC is part of fitness function calculation. In the context of our encoding it is done in two steps. First for each parallel block a matrix representing the part of the circuit is calculated. Then each such a unitary matrix is multiplied with its neighbor so as at the end a matrix representing the whole circuit is obtained. An example of this calculation can be seen in Figure 15. Inside of each parallel block only Kronecker matrix products are used, while standard matrix multiplications are used to multiply these blocks.

In the case of our software variant for RC synthesis the evaluation is similar to the one used in QC but no matrix operations are required and truth tables of (in)complete multi-output functions are used instead. As mentioned earlier, constants insertion is possible in RC logic synthesis. Consequently a function defined over 3 wires can be searched on a circuit with 4, 5 or even more wires. The evaluation of a RC circuit consists in finding logical equation for each wire and then testing all input/output combinations on the searched wires. In this case the constants are alternatively tested for each wire.

As can be seen, the length of the chromosome is proportional to the size of the circuit. Also the time required to calculate each circuit is proportional to this size. A circuit with seven gates will have seven blocks if each gate cannot be connected in parallel with another one. In such case, the chromosome with seven blocks encoding a circuit for 3 qubits can encode maximally 7*3 = 21 one-qubit gates.

8. The Fitness Function

The fitness function used was multiple times modified in order to observe improvements or changes in the search for the new optimized circuits. For dydactic reasons we will explain some of these variants. The fitness function evaluates each individual and assigns to it a fitness value, representing the quality of the encoded circuit, and the encoding is a direct mapping from genotype to phenotype. The original GA was defined as GA with Darwinian learning. It means, the fitness function evaluates the genotype. In Baldwinian GA the genotype is converted to the phenotype and the fitness function evaluates the phenotype. The objective of our experiments was consequently to modify the fitness function in order to observe the impact on the "evolution" of the desired solution. The fitness function does not modify the circuits but only biases the selection operator. This mechanism, according to the fitness function value of each circuit, will be able to select better or worse circuits. For example, a fitness function weighted too strongly towards the length of a circuit will prefer circuits with higher length, even if they are not correct. This kind of flexibility allows the GA to explore regions of the problem space that are inaccessible to classical computational methods without an extreme time and computational resources consumption.

The equation (7.1) represents the basic fitness function.

$$F = \frac{1}{1 + Error} \qquad (7.1)$$

where Error is the evaluation of the correctness of the circuit. Although we found this fitness function very useful in space exploration, it is however too generic and does not take into account the cost of the circuit. An improved function is shown in equation 7.2. As can be seen in the equation, two parameters are sufficient to create a more sophisticated fitness function

$$F = \frac{1}{1 + Error} - \frac{Min\_cost}{Cost} \qquad (7.2)$$

where the first right hand element is the evaluation of the correctness of the solution from equation (7.1) and the next one is an additional constraint forcing the selection operator to select circuits with a smaller number of gates or with gates having a smaller total cost. In the case of the QC search the error evaluation is based on the comparison of the resultant matrix with the unitary matrix of the evaluated circuit. The equation (7.3) describes the Error calculation

$$Error = \sum_{i} \sum_{j} \left| O_{ij} - S_{ij} \right| \qquad (7.3)$$

where O_{xy} is an element from the expected synthesized unitary matrix and S_{xy} is an element from the matrix of the current circuit (for speed a truth table is used in case of RL). The error generated by this function is exact because it reflects any differences in the whole matrix of the circuit. In the case of an exact match, this error substituted in the equation (7.2) gives a fitness of 1. As can be seen in Table 6, the error has always value 128 in the denominator, since there are 64 (2^3x 2^3) units in the evaluation matrix of a 3-qubit gate and each of them is a complex number. The representation of complex numbers in computers is done via a definition of complex numbers as one number for the real part and one for the imaginary. Consequently one can scale the error either versus the complete set of real and imaginary parts of the matrix coefficients either versus the number of coefficients in the matrix of the resulting circuit.

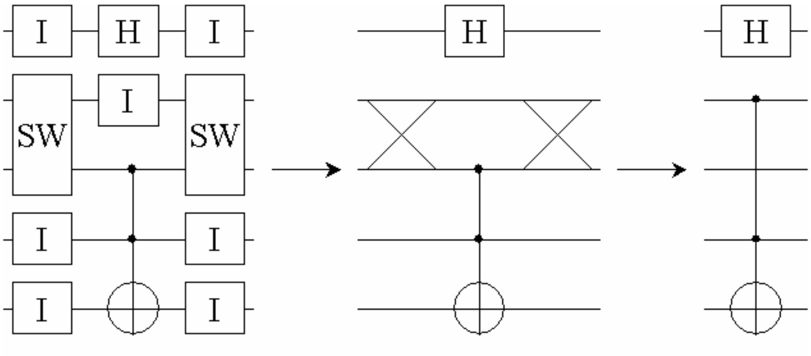


Figure 14: Transformation of a QC from the encoded chromosome (on the left) to a final quantum circuit notation representation of this QC (on the right). Here S is a Swap gate, H is a Hadamard gate and W is a wire. In the middle there is one CCNOT (Toffoli) gate.

Although useful in several runs of our program, the equation (7.2) has still a slight imperfection. In the case one wants to measure the fitness in the interval [0, 1] the second element of the equation (7.2) will in some marginal cases output a fitness of 0 while the correct result was found. Consequently a modification of the fitness function is required. Equation (7.4) shows one possible form of a function producing fitness in the interval [0, 1].

$$F = \frac{1}{Error + \frac{Min\_\cos t}{Cost}} \qquad (7.4)$$

However, this equation assumes that Min_Cost is the cheapest possible circuit. In the case a cheaper correct circuit is found the fitness will be of course smaller than 1.

This fitness function is centered in the point of the optimal circuit (Min_Cost and correct result) similarly to a Gaussian curve. It is clear from this example that the Min_Cost parameter should be also set to 1 which leads to the next version of the fitness function from equation (7.5):

$$F = \frac{1}{Error + Cost} \qquad (7.5)$$

This equation then should be only maximized because the value of 1 is practically impossible to find. It is subsumed in the Cost parameter. The value of used gates varies as previously introduced in this paper. Consequently a circuit realized with only 1-qubit gates will always have higher fitness than one realized with gates of two and more qubits if for each more-than-one-qubit gate a cheaper variant with only 1-qubit gates is found. Important point to notice is the numerical non-compatibility of the Error and Length parameters. The Error is in the range [0, $2^{1+(2*n)}$] while the range of the Length parameter is in [Min, ∞]. This ill-scaled ratio would in consequence make the selection pressure mainly with respect to the length of the circuit and not to its correctness. As a solution to this problem, the final variant of the fitness function scales these two variables as shown in equation (7.6):

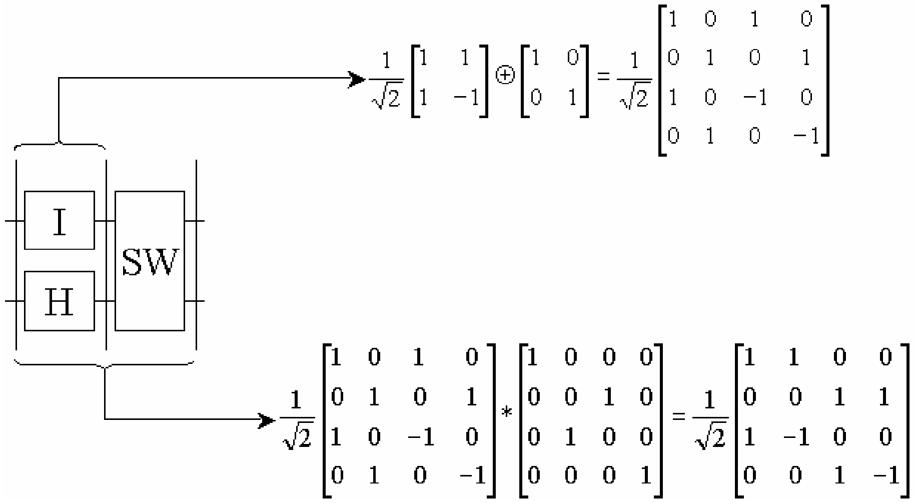
$$F = \alpha(1 - \frac{Error}{Max\_error}) + \beta\frac{1}{Cost} \qquad (7.6)$$

where α and β are scaling parameters that allow to modify the selection pressure either towards cheaper or more correct circuits.

Another possible approach induced from the previous fitness function was the introduction of the Pareto optimality (PO) fitness function [9]. PO is used for multiple objectives optimization problems such as here the Cost and the Error. A GA using PO uses a different evaluation for the fitness of each individual. This approach is the following:

- Each individual is compared with all others and is assigned with a rank.
- The rank of an individual depends on the number of wins while compared to all other individuals.
- The comparison is made on all parameters. Example: an individual with the highest rank will have all of its parameters higher (for maximization) or lower (for minimization) than most of all other individuals.

Such a comparison forces the selection process of the GA to reproduce mainly the globally optimal individuals. Similarly to equation (7.5) one needs to scale parameters of comparison when they are not equivalent.



$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\oplus\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}=\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\\ 1 & 0 & -1 & 0\\ 0 & 1 & 0 & -1\end{bmatrix}$$

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\\ 1 & 0 & -1 & 0\\ 0 & 1 & 0 & -1\end{bmatrix}*\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}=\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1 & 0 & 0\\ 0 & 0 & 1 & 1\\ 1 & -1 & 0 & 0\\ 0 & 0 & 1 & -1\end{bmatrix}$$

Figure 15: Examples of Kronecker product ⊕, and of Matrix product * on a sample of a circuit.

9. Other parameters of GA

In the following discussion of our results, Stochastic Universal Sampling (SUS) was used as the selection operator, although we experimented also with Roulette Wheel and other selection approaches. A decision for using SUS instead of Roulette Wheel (RW) was implied from the following demonstration of high non-linearity in circuit design. Consequently an operator allowing a selection of less locally optimal circuits such as SUS was needed in order to obtain good experimental results.

Circuit (blocks)	Error (3)	Cost ΣC_g	Fitness (6) $\alpha = 0.9$ $\beta = 0.1$
1	5/128	5	0.8815
2, 3, 4	18/128	12, 18, 24	0.78, 0.778, 0.777
5, 6	4/128	30, 36	0.875, 0.874
7 (circuit completed)	0/128	42	1

Table 6: Results of sequential evaluation of Fredkin gate from Figure 5a.

An implication of the previous part of this paper, one of the goals of the presented work was to “reinvent” gates such as those in Figures 2, 3 and 4, or to synthesize smaller ones with the same functionality. Consequently, the analysis of a circuit can give more precise explanation of possible non-monotonicity in the synthesis process evaluation. For this purpose let's have a closer look at the circuit from Figure 8a. This circuit has seven gates. The measurement of Error as from equation (7.3) and fitness from equation (7.6) are recorded in columns of Table 6.

A closer look at the above table will reveal non-monotonicity in the evaluation of this circuit. The first "trap" for GA is between the first gate and the adding of the second. Because both the error and the cost increase, the fitness is significantly reduced and creates a relatively big local minimum. Consequently this step is difficult for any automated synthesis without heuristics. Two next "smaller" problems are that between steps 2, 3, 4 and 5, 6 the error remains constant while the cost of the circuit is linearly growing with each segment. A monotonic growth of the cost can be observed in the third column. It is important to notice that all gates have a cost including the wire gate (identity gate). Such a setting allows the algorithm to avoid some local minima and constructing a circuit with only empty wires. From the above example an important modification to the algorithm can be derived and is discussed below.

A solution to this non-monotonicity problem can be a modification of the above presented genetic operators. In this work an extension of the mutation operator was created. Its result is the addition of a complete parallel block to the circuit. Also the mutation is tested for each gate of the circuit because multiple mutations can occur on one chromosome. Such a "bitwise" mutation or noise introduction into the circuit can avoid such local minima as those in Table 6. The modification of the operator is required to override such traps as in Table 6. Moreover, as already mentioned, in each of the groups of steps {2, 3, 4} and {5, 6} the cost increases while the fitness decreases. It is possible to use a fitness function such as (7.1), however then the solution can have an arbitrary length and no parameters force the selection of a shorter circuit. The results and their comparison is described in the next section.

10. Experimental results

As was shown earlier, several well-known gates can be synthesized in different manners depending on the cost function selection. There exist already a variety of implementations of universal gates, especially Fredkin or Toffoli. Our goal, however, is to obtain the gates of the smallest possible cost for realistic physical realizations. In section 2 of this paper few variations of Toffoli and Fredkin gates were given and we discussed possible minimization of their cost by realizing them with smaller, truly quantum, primitives. The two important properties allowing the minimization operation are the commutability and concatenation properties of gates in QL. The first property, whether order of two subsequent gates can be changed, is purely mathematical and can be studied in any literature on Quantum Computing. In general AB ≠ BA, but some operators (gates) do commute. This can be used in synthesis, for instance as part of local optimization algorithms from section 6. The second property is more problematic because there are no any theoretical basis allowing or not this operation to be applied. It is only a heuristic useful to calculate approximate cost functions of composed gates [21] and the issue of concatenation should be further investigated.

All runs of the GA have these generic settings. The mutation probability was in the range of [0.01,0.1], the crossover probability was in interval [0.75, 0.9], and the size of the population was [100, 200] individuals. All runs were done on a Pentium 500Mhz with 64 megs of ram.

We reiterate again the non-monotonicity issue discussed previously. Because of two contrary flows of values in the evaluation function (growing a cost and reducing an error) the modifications to the fitness function and selection operator needed to be done. The first possible and already introduced modification is the scaling of the fitness function with the coefficients α and β. Such a use of scaling coefficients allows to eliminate the too small negative modifications due to the linearly increasing cost of the circuit. It is advised and experimentally proven in following results that the α coefficient should be in the interval of 0.9 to 0.99 while β should be a complement to it; $1 - \alpha$. Similarly these coefficients should be used in a PO selection operator to scale the ranking of individuals.

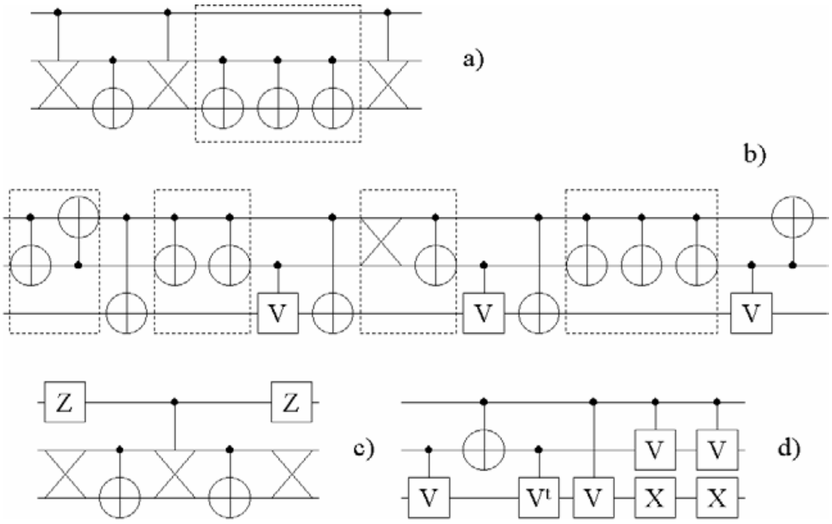


Figure 16: Various realizations of Toffoli gate obtained from GA before optimizing transformations.

Before comparing available results one needs to define the input gates for the GA. In this case a distinction between a complete starting set and a biased starting set has to be made. A complete starting set of gates is one containing all available gates for synthesis, excluding the resulting gate. In a complete starting set of gates the following gates were available: Wire, Hadamard, Pauli X (NOT), Pauli Y, Pauli Z, Phase, V, Feynman, Swap, Controlled-V (C-V), Controlled-V-Hermitian (C-V^+), Fredkin, Toffoli, and Margolus. From these the resulting one was removed. A biased starting set of gates is one based on some constraints, for instance only 2-qubit gates are used, or only an arbitrary subset of gates is available. In figures representing results of the search following letters represent these gates: Z – Pauli Z, V – controlled V, V^+- controlled V^+.

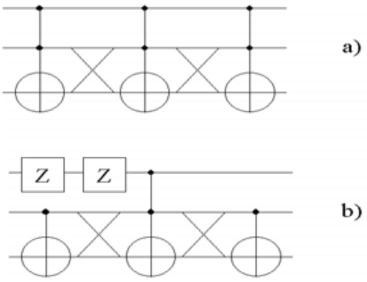


Figure 17: Results of synthesis of Fredkin gate. a) is result of application of fitness 7.1 and a complete set and b) the result of fitness 7.6 and a complete set.

Let us compare results of different runs using various fitness functions. Figure 16 presents the results of search for the Toffoli gate. The first circuit (Figure 16a) is the result of a run where fitness from equation 7.1 was used and a complete starting set of gates. As can be seen, this result is very expensive because includes three Fredkin gates and four Feynman gates. This circuit can be simplified by removing two of the three subsequent identical Feynman gates based on the fact that Feynman gate is its own inverse. The next one, Figure 16b, is a result of the same setting as with the previous one but the starting set was a biased one. The available gates in this biased set were only the ones with number of inputs smaller or equal to 2 qubits. Similarly to the previous circuit, the group of gates in dash-squares can be concatenated in order to reduce the cost of the circuit. Moreover two pairs of consecutive identical Feynman gates (in dotted groups) can be removed. Thus, the second from left dotted group is removed entirely and the first from right dotted group is replaced by a single Feynman gate. The rightmost Feynman gate can be moved before the last Controlled-V gate and

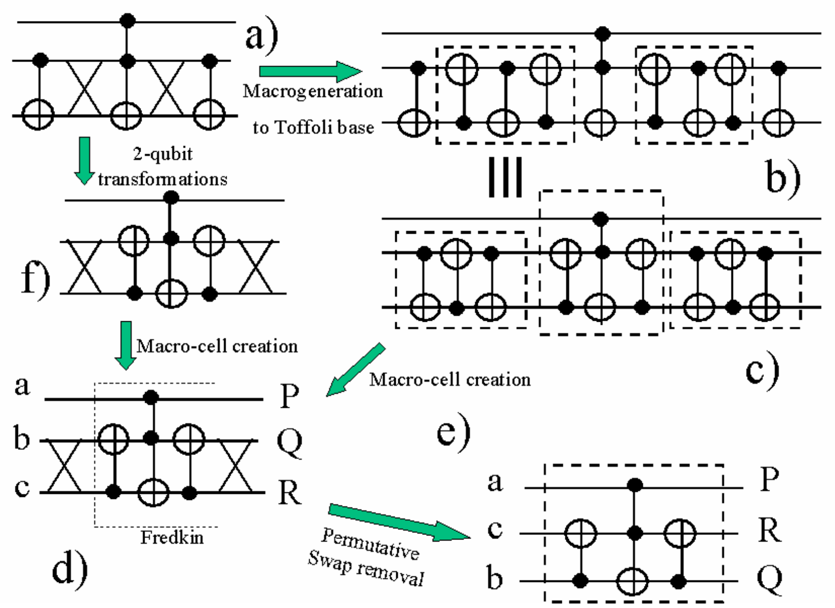


Figure 18. Simplification of Fredkin gate based on local transformations: (a) the gate found by the GA, (c) result of macro-generation to Toffoli Base, (d) circuit from Fig.18b rewritten to show left-side patterns, (d) result of macro-cell creation, (e) final circuit after permutative Swap removals, (f) another result of 2-qubit transformations in Toffoli Base applied to the initial circuit.

added to the dotted rectangle. Although this circuit is longer than the first one its cost is the same. It is composed only from 2-qubit or 1-qubit gates. The first circuit has cost 55 (15*3 + 5*2) after minimization and the second circuit also after minimization has cost of 45 (9*5). The two circuits below in Figure 16 were found using the improved fitness function from equation 7.6. The circuit from Fig. 16c was found using a complete starting set and the one from Figure 16c using the biased set similar to the one used for the circuit from Figure 16b. The major improvement using the equation 7.6 as a fitness function is that the results are reproducible, because they are driven as already mentioned by two opposite flows of evaluation. Observe that by flipping over two Feynman gates from Figure 16c and removing the swap gates a solution with one Fredkin and two Feynman is used, which is close to a known realization of Toffoli gate from Fredkin gate. The only difference are two Pauli-Z rotation gates, which can be removed, as analysis shows. Interestingly, our software reinvented also the famous circuit of Smolin, since the sequence of two Pauli-X (NOT gates) can be removed, and the sequence of two Controlled-V is equivalent to

Feynman gate. After these transformations, our gate is composed from the same basic quantum primitives as the Smolin's solution, but in a different order. The analysis of unitary matrices shows however that the circuits are equivalent. These examples show that not only can our software "reinvent" the realization of the known gates but it can also create similar minimum cost realizations of other gates, as will be presented below. We were not able to find a better solution to Toffoli and Fredkin gates from quantum primitives, since perhaps they do not exist.

Figure 17 presents two of many found realizations of the Fredkin gate. Again, observe that the consecutive Pauli-Z gates can be cancelled since this gate is its own inverse (a standard local transformation). Next, two Feynman gates can be flipped over and corresponding swap gates removed, leading to the known (minimal) realization Fredkin gate using one Toffoli and two Feynman gates. Figure 18a shows step-by-step simplification of the circuit from Figure 17b after removal of Pauli-Z gates. Applying macrogeneration of gates to Toffoli Base the circuit from Figure 18b is created which is the same as one from Figure 18c, in which gates were differently grouped to satisfy the left-hand-side patterns of rules. Then macro-cell creation leads to the schematics from Figure 18d. Finally, permutative transforms from set S13 that remove Swap gates lead to the classical realization of Fredkin gate from Figure 18e. Observe that the same solution is found when 2-qubit transformations are applied to Figure 18a (leading to the circuit from Figure 18f) and next the macro-cell generation is applied to the array from Figure 18f.

The circuit from Figure 17a is the result of using the equation 7.1 and a complete starting set. Fig. 17b is the result of fitness equation 7.6 and the same starting set as in Fig. 17a. The realization of the Margolus gate from Figure 19a is elegant and new. By applying the swap-related transformations of the Fredkin gates (Figure 19b, Figure 19c) two circuits with two Fredkin gates each are created that are not known from the literature. The 3-qubit Kerntopf gate family includes gates with one Shannon expansion (of the form v'u+vx, where v,u,x ∈{a,b,c,a',b',c'}and the expression cannot be simplified) and two Davio expansions (these expansions have the form v ⊕ ux, where v,u,x ∈{a,b,c,a',b',c'}and the expression cannot be

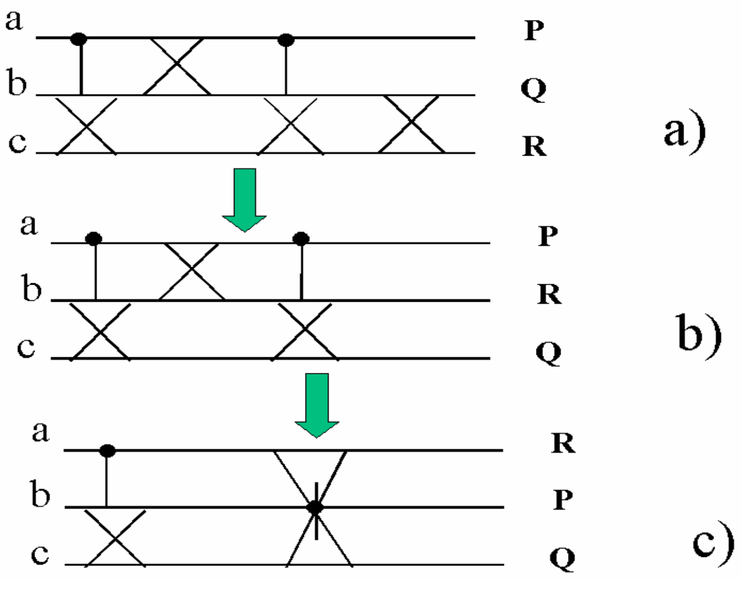


Figure 19. Margolus gate: (a) Original gate obtained directly from GA, (b) Removal of the right Swap gate permuted the order of output wires Q and R, (c) removal of the first Swap gate permuted output wires P and R.

simplified). Figure 20a presents a gate from this family in Toffoli Base, and Figure 20b shows a gate from the Margolus gate family (NPN class), also realized in Toffoli Base. Based on these and other results, we can claim that our program invented several realizations of basic gates that have been not created yet by humans and are perhaps patentable.

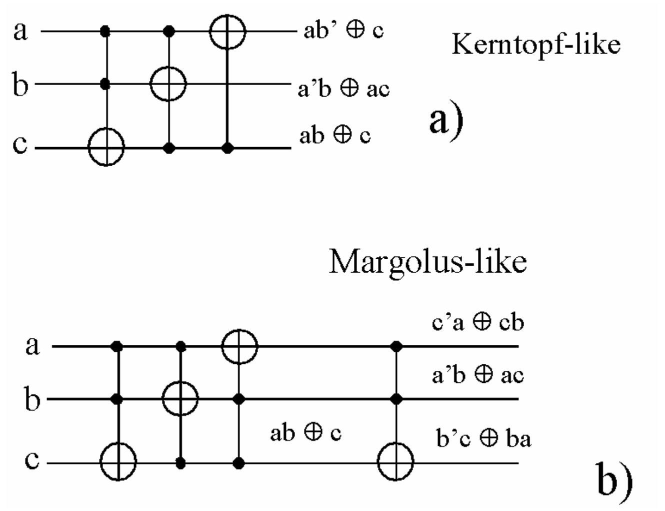


Figure 20. Kerntopf-like and Margolus-like gates in Toffoli Base after optimizing transformations. These are the best reported results for such gates.

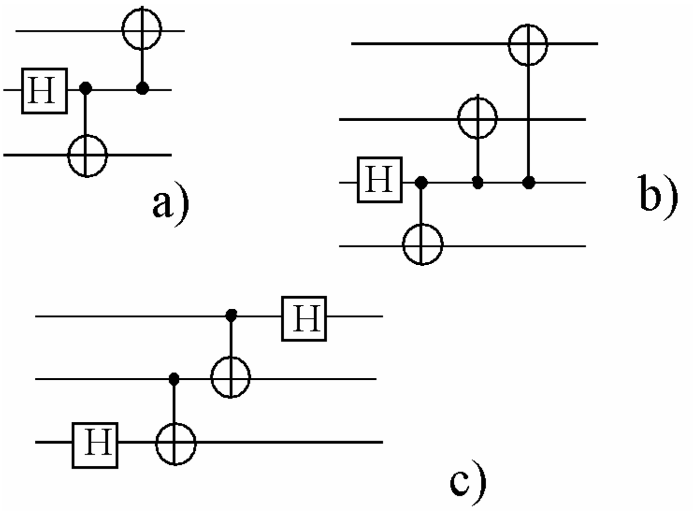


Figure 21. Entanglement and teleportation circuits from the literature found also by our program.

Figure 21 shows the entanglement and teleportation circuits found also by the GP approach from [5]. Finally, Figure 22 illustrates the using of transformations to prove that Fredkin gate is its own inverse. Transformations of this type lead often to significant reductions of long arrays found intially by the GA.

One of the striking observations of the obtained results is the respective presence in Fredkin or Toffoli gates in the synthesis of each of them. As can be seen in the results of synthesis using the complete starting set, the Fredkin gate is present in all Toffoli implementations and vice versa. It can be concluded that there is a local minimum that our algorithm was unable to override. Overriding this local minimum can be done using only basic quantum primitives such as controlled-V.

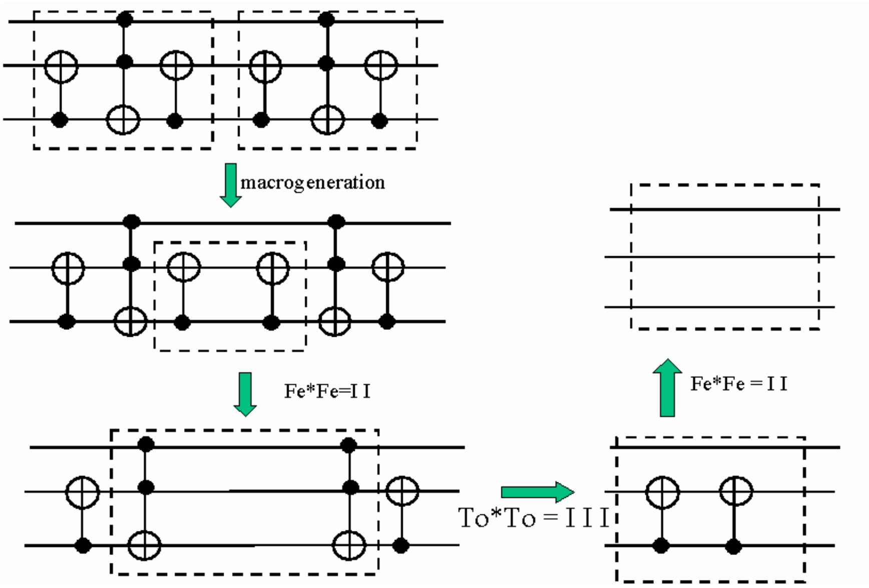


Figure 22. Using simplifying transformations to prove that Fredkin gate is its own inverse.

An interesting difference between our results of synthesis of Fredkin or Toffoli gate and those shown in Figures 5 and 6, is that our algorithm found circuits with generally slightly higher cost (in the case of improved fitness function). To understand this point one needs to look at the evolution of the search during a run of a GA. Figure 23 shows the recording of one run searching for a gate using the fitness function 7.1. For illustration the fitness 7.6 is also drawn. The figure shows the best result of each hundred generations. First curve Error shows the evolution of the scaled error as used in the fitness function. Second curve is the cost (1/Cost) and it is to be maximized to increase the global fitness function. The larger a circuit the more the cost is reduced. The last three curves show the values for fitness from equation 7.1 and 7.6. Fitness function 7.6 is shown by two curves; first with parameters α = 0.99 and β = 0.01 then with α = 0.9 and β = 0.1. Moreover for more clarity all fitness functions are mapped on the secondary y-axis on the left. Since there are many solutions to a gate at a higher cost, it is more likely that GA finds those solutions, but hopefully running the local optimizing transformations on each of these non-minimal solutions leads to the same optimal solution, as was the case with the solutions discussed above.

Figure 23 illustrates the problem of fitness function used in quantum logic synthesis. While both variants of the fitness function are globally stable or stuck in a local minimum, the parameter having the greatest variation is the cost of the circuit. This is because, as mentioned before, an infinity of circuits exist theoretically in QL for a certain unitary matrix. This is illustrated by the fact that while the error is constant all along the run, the cost oscillates. Moreover, once the correct circuit is found, the fitness function 7.1 is at its maximum but the fitness function 7.6 is relatively smaller because the circuit is larger than the local minima shown in the graph. For this reason while using the fitness function 7.6, once a correct circuit was found its fitness was set up to 1 and the run was over. The presented curves capture also different levels of detail while searching for a correct circuit. The Fitness function (fitness-7.6 (0.9)) captures strongly the size of the circuit but the final jump to the correct solution is very large in the value of fitness (≈0.1). Such a fitness function can be used to minimize the size constraints. However, the best fitness function to capture global properties during a search for a circuit is the fitness function (fitness-7.6 (0.99)), because on one hand it is mainly influenced by the correctness of the circuit and on the other hand it takes into account the size of it. Consequently, comparing it to the fitness function (fitness-7.1) one can observe similar result but with small oscillations in the fitness function (fitness-7.6 (0.99)).

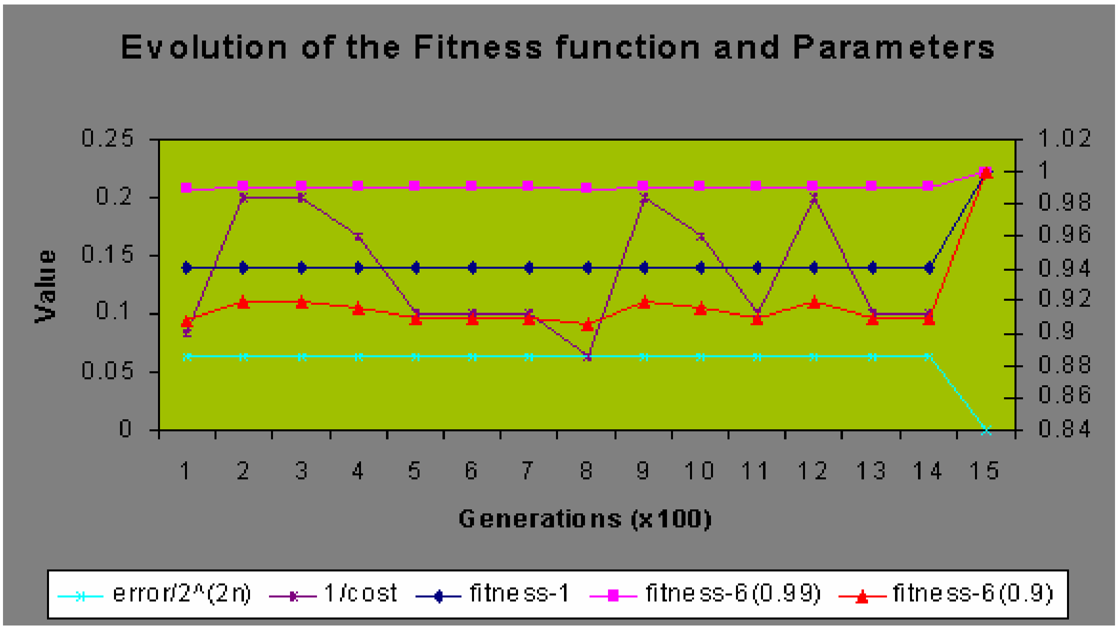


Figure 23: An example of evolution of the fitness function and its related parameters during a run. For more precision the two fitness function curves are shown on a secondary y-axis on the left.

An important point to be discussed is the improvement of the results obtained with the hereby proposed cost functions and fitness functions variants. Intuitively one could think that the cost of gates reducing the global cost of the solution will only improve the results. However, as can be seen in Table 6 and Figure 23 the reasoning is not so simple. The duration of runs using normal fitness function as in equation 7.1 has generated solutions in less than 2000 generations. These results were not optimal; either too long and too expensive, or not correct at all. On the other hand the parameter that forces the selection operator to pick the individuals with smaller gate costs drives the evolutionary process deeper into the problem space. As the consequence, the search time using the improved fitness function increases. Also, when using the fitness function 7.6, the parameters α and β can be set to values in the interval [0, 1]. Ultimately when using α = 1 and β = 1 - α, this leads to similar results as using the fitness 7.1. The problem remains the same: *"What are the correct parameters for an optimal relation between the error evaluation and the cost of the circuit in order to find the optimal representation of the searched function?"*

Problem searched	Solution found F7.1/F7.6	Time of search in number of generations F7.1/F7.6
Toffoli	YES/YES	<2000 / <50000
Fredkin	YES/YES	<1000 / <75000
Margolus	YES/NO	<1000 / <100000

Table 7: Comparison of Fitness Function variants

Table 7 presents a summary of some results in QC search. The first column is the function searched, the second column shows if the solution was found using either the non-optimized fitness function (F7.1), or the optimized one (F7.6). The last column shows, for each circuit, the time in generations necessary to arrive at a solution. Each run was stopped after 100000 generations in the case no solution was found previously in the run. The above presented argument that the improved fitness function increases the required time to find a solution can be argued here. As can be seen in two cases out of the three problems, a significant time increase was observed. The presented results are statistical averages over five runs for each problem. Out of them only the best solutions are presented in Figures 16, 17, and 19. Two arguments support the use of the improved fitness function. First, the solution was found in all cases and it was found even when using a biased set. This implies that the cost function allows the GA to explore parts of the problem space that were not accessible to the GA which used the normal fitness function, i.e. without the cost of gates. Also all results from Table 7 are from runs using the biased starting set of gates.

The following observations can be made when using the GA proposed here with variable length of chromosomes for quantum logic synthesis.

- No direct relation between the error and the cost values was observed. In general this means that there is almost an infinity of solutions to a given problem and the number of these solutions grows with increasing size of the circuit. A solution of non-minimal length can be found easier because of the variable length of chromosomes and next transformed to a shorter solution. It should be further investigated if this a better strategy than restricting a size to given value. When the length is short, such as 5, the second strategy works, but for longer circuits the variable length of chromosomes seem to be more powerful.
- When drawing the fitness of the best individual each (let's say 100th) generation we observe that there is no tendency of optimization (convergence). This confirms our previous conclusion that the space of the fitness function is non-monotonic with lot of flat space (local optima) and some occasional peaks of good solutions.
- The best type of Mutation is Bitwise, and the best type of Replication is SUS. The bitwise mutation operator seems to be more appropriate for such a high noise space as the one explored here. Also, this observation is based on an average measurement of fitness taken from the best individuals from each 100th generation but this approach does not necessary finds better solutions.
- Best type of GA observed from all previous results is Baldwinian, using SUS and bitwise mutation operator.

To illustrate these mentioned problems and conclusions we set up a small comparative experiment. In this experiment, three different parameterized GA were used. Two types of comparative experiments are set up and are used to explore the fitness landscape. In the first case all three configurations are using the same value of parameters (mutation, crossover, etc.) and five different parametric settings are modified. For this a binary encoding of each GA type was used. Each GA has a binary signature such as 00000 or 10110 corresponding to a particular configuration. Each digit in the binary signature represents if a certain feature is used or not. The five positions in the signature correspond respectively to the following options: fitness type, threshold, replication type, mutation type and Baldwinian. Fitness type means that if it is set to zero, the fitness will be calculated individually and if it is set to one the fitness will be calculated as a shared fitness in a group of 20 individuals. Threshold set to one means only individuals having fitness higher than threshold (0.6) will be used for replication. Replication type set to zero means the GA will be using the Roulette Wheel type of selection and if it is set to one the stochastic universal sampling is used. The mutation type set to zero controls the GA to use normal mutation (once pre individual based on the mutation probability) while set to 1 selects the bitwise mutation (applied to each element of the chromosome). And finally Baldwinian bit indicates if the GA is using fitness calculated directly from the chromosome or calculated from a minimized chromosome. The results are presented on Figure 24. As can be seen the GA behaves differently for different parameters. In the case where no fitness = 1 is attained the GA was not always successful in finding the solution, while in the opposite case the GA found a good solution in at least in half

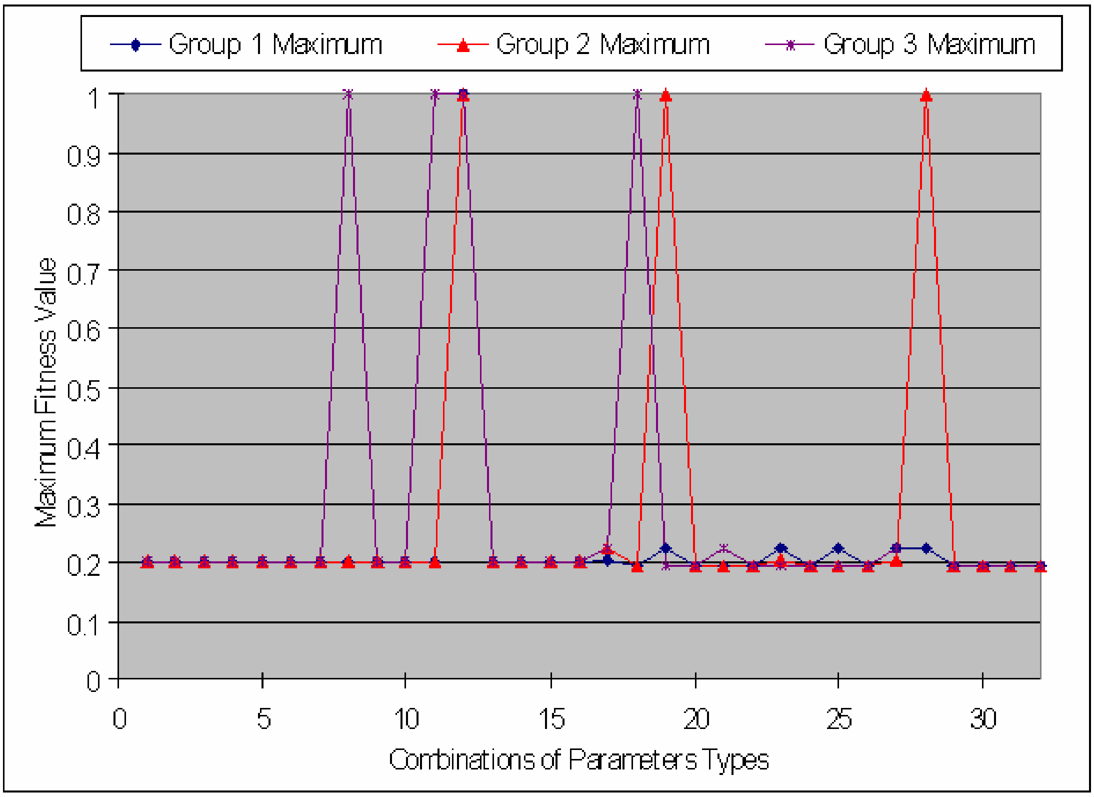


Figure 24: Results of the comparative tests with all 32 possible variations of the GA. Results are all normalized over 20 runs for each. The X axis represents each of the 32 possible combinations of the 5 concerned parameters. Y axis is the fitness function as from equation 7.6.

of runs. Each point on the figure is a result of a statistical mean over 20 runs. The main result form this experimentation is the fact that some configuration of the GA is more successful than others for the quantum search space search. This concerns mainly GAs of types 01000, 01011, 01100 and 11100. These configurations were experimental proofs of better adaptation and better search ability than others.

The second type of experiments comparing different GA settings is also based on three groups. In this approach the settings from Table 8 were used.

Population Size	50-70-100	Calculation of fitness	Individual
Mutation Probability	Variable [0.01 - 0.3]	Type of mutation	Normal
Crossover Probability	Variable [0.2 - 0.9]	Type of crossover	1 point
Factor Alpha	Variable [0 - 1]	Type of replication	RW
Factor Beta	Variable [0 - 1]	Type of fitness	Complex

Table 8: Options and Parameters (Group 1)

Three GA were created; each with 50, 70 and 100 individuals in the population. Each GA was configured as follows: no Baldwinian fitness calculation, normal (once per individual) mutation, single point crossover and individual fitness calculation were used. The parameters to be varied in this approach were: Alpha, Beta, mutation and crossover probability. Only a resume of the characteristics of Alpha and Beta factors and their best, worst and average values are listed in Table. 9. We have chosen four sets of Alpha and Beta parameters, they are Alpha=0.7 Beta=0.3, Alpha=0.8 Beta=0.2, Alpha=0.9 Beta=0.1, and Alpha=0.97 Beta=0.03. Note that the relation between Alpha and Beta is that Alpha + Beta = 1.0.

We observe from the results in Table 9 that the fitness value increases when the Alpha value is increased. However, the best fitness of all generations is with the smallest from the proposed Alpha values, thus there is a two-fold (advantage and drawback) characteristic in increasing the factor Alpha. This confirms again the fact that the relation between the measure of error and the circuit cost is not direct. Based on solution analysis, it indicates that the size of the circuit should be controlled in a more direct way and not to be left to the evolution alone.

F	Group 1			Group 2			Group 3		
α	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
0.7	0.44	0.0777778	0.148206074	0.377778	0.0636364	0.144381594	0.188142	0.0333704	0.141694126
0.8	0.163663	0.0727273	0.159274043	0.163663	0.0652016	0.160634195	0.272727	0.0615385	0.161248625
0.9	0.181832	0.18	0.18007352	0.28	0.18	0.18110992	0.216788	0.18	0.18144116
0.97	0.224	0.194	0.19431102	0.194549	0.194	0.19403843	0.194549	0.194	0.19403294

Table 9. Characteristic of Fitness *F* with relation to Factors Alpha and Beta

Next we analyzed also the influences of the Probabilities of Mutation and Crossover on the Fitness Value. Again, only a resume of the characteristics of

Mutation and Crossover Probabilities and their best, worst and average values are to be listed in Tables 10 and 11, respectively.

F	Group 1			Group 2			Group 3		
MP	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
0.01	0.194549	0.194	0.19401098	0.194549	0.194	0.19401098	0.194549	0.194	0.19401098
0.05	0.224	0.194	0.19431102	0.194549	0.194	0.19403843	0.194549	0.194	0.19403294
0.1	0.224	0.194	0.19503785	0.224	0.194	0.19491647	0.224	0.194	0.19432749
0.3	0.224	0.194	0.19491651	1	0.194	0.19491651	0.224	0.194	0.19465308

Table 10 Characteristic of Fitness *F* with relation to Mutation Probability (*MP*)

F	Group 1			Group 2			Group 3		
CP	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
0.5	0.194549	0.194	0.19401098	0.194549	0.194	0.19401647	0.224	0.194	0.19491647
0.6	0.224	0.194	0.19403294	0.194549	0.194	0.19402749	0.194549	0.194	0.19403843
0.8	0.224	0.194	0.19432749	0.194549	0.194	0.19403843	0.194549	0.194	0.19403294
0.9	0.194549	0.194	0.19465308	0.194549	0.194	0.19402749	0.194549	0.194	0.19402196

Table 11 Characteristic of Fitness *F* with relation to Crossover Probability (*CP*)

Comparing the obtained results from Tables 9, 10 and 11, we arrive at the following conclusion: the quantum circuit synthesis is not extremely dependent on the values of such parameters as mutation or crossover probability. Rather, it depends on the types of selected operations on chromosomes that are used in various GA variants in experiments. This means the structure of the quantum space is not sensitive to the parameterization of genetic operators but is sensitive to the operations themselves. This is mainly because the size of the circuit is variable and the space of possible solutions available before and after one genetic operator is applied changes radically. In other words this means that for example having a circuit with 6 blocks that will have a group of solutions based on first four blocks equal to A, B, C, D with two additional blocks that are required. Now application of the mutation operator that will remove one block will completely modify the current outlook for success because it can be assumed that the same sequence of first four blocks A, B, C, D does not lead to a solution with 5 blocks. Consequently, the changes of length in the chromosome of individuals in the GA are one constraint more for the GA to overcome.

The attentive reader will remark the Pareto optimal GA results were not analysed. This results from a weak relation and non-equivalence of the two main parameters of our fitness function; the error and the cost. The Pareto optimal GA can be applied in the case the solution is on the intersection of best results of all parameters evaluation. This is not the case here because the cost of the circuit will never be minimal. A circuit representing a function will always have a cost equal to at least one parallel block with gates. Moreover, because in the ranking of individuals the error and cost are used to determine a win or a loss, the algorithm has the tendency to fall too soon into a local minimum with a relatively small error and a very small cost, from which there is no escape.

11. Important Research Issues.

The results of this work proved once again the “no free lunch theorem”. The modifications of some parameters results in improvement in some runs but do not provide optimal solutions. Consequently the optimization of any algorithm needs to be specific to the proposed task. In our case, we have shown that the GA is well situated to explore the problem space of RC and QC logic synthesis. However few heuristics have been added to our algorithm and they provide us with a fresh look at this new area of logic synthesis.

An important issue to be discussed is the representation and transformations of circuits with more than 3 qubits. In the presented work we focused on 3-qubit universal gates. Consequently, the number of possible permutations of used gates is finite and a sufficient number of them can be defined in the starting set of gates. The definition of larger elementary units can be directly implied from the design of our individuals in the GA. As each circuit can be partitioned into parallel blocks, multiple optimizations can be used to improve the results. An example is to define blocks of predefined size bigger than two qubits, called complex blocks (CB). Consequently multi-gate blocks will be used as simple gates and will be manipulated by the classical genetic operators. One of such blocks can be seen in Figure 14. It was also illustrated in Figure 14 that synthesis of Toffoli on wires 2, 4 and 5 was too complicated to letting the GA to find such a gate. Particularly in QC there are many examples where more than 1-qubit gates are required to be connected to non-adjacent wires or even the wire order needs to be completely inverted. Consequently, our approach to solve this problem that evolved in the course of this research was to insert larger elementary units than simple gates. An example is the creation of a Feynman gate on wires 1 and 3. It can be argued that there are very many gates that could be defined such as this one, so should we define them all? The counter argument is that most gates can be synthesized from gates already used in our work. In our approach, the gate is connected to wires according to its position in the parallel block so the implementation of genetic operators is easy and can be executed quickly. The number of defined gates relates also closely to the number of respective equivalence transformations and the efficiency of rule-based optimizer decreases with too many rules. We believe that a middle way is the best one – there should be the major quantum primitives and basic gates defined on all wires and on all permutations of inputs. The penalty we pay however, is a big number of stored unitary matrices of basic gates. There is a solution though – declaring just subsets of available gates for every run. Thus only some of permutations of gates like Toffoli are declared in each run. With high speed of generating solutions and relatively high number of solutions for each matrix, our approach is competitive and allows various types of experimentations.

The approach presented here can be compared to a GA where each gate can be defined over a set of wires, with control bits and other parameters [5]. In that case the mutation operator needs to be redefined to be able to modify any of the gate parameters. Consequently the number of possible variations of one gate grows quickly with the number of parameters of this gate and the number of wires it can be connected to. This makes genetic operators and evaluations of matrices in fitness function more complicated and slower. In our method all these parameters disappear and are replaced by the position of the gate itself in the parallel block. The negative point is that we need to define a particular gate for most of the variations of one gate in order to speed up the search. For 3-qubit and 4-qubit gates our approach is more efficient than having a highly complex parameterized gates such as in [5]. A related important problem is how the GA representation affects the speed of fitness function calculations for larger circuits. The critical factor are definitely the calculations of resultant unitary matrices in fitness function, and especially the Kronecker matrix multiplications. In one approach, which is more flexible, the matrices are created in a run by decoding the GA or GP operators. In another approach, used by us, all matrices are calculated in advance and stored in the memory. They correspond to characters used in chromosomes and do not need to be created in a run. It results from our experiments that this approach is faster for small circuits since the gain of flexibility of the other approach is practically useless as leading to a too large time increase. On the other hand our approach does not scale well for a higher number of qubits than 5, because it requires to define and store many matrices. So, although more efficient than previous approaches, our encoding will not scale well for larger circuits and further research on synthesis of such circuits is necessary.

Another useful idea related to gate and circuit design that we acquired in the process of this work is the following. As can be seen from the examples, sometimes the solution can be found which is a permutation of output wires with respect to the initial specification. Because in many problems the order of output wires is not important, there should be some mechanism in the software that would automatically perform simplifications not to the circuit described by original order of output wires but to some of its permutations. A theory of respective local transformations should be developed to solve this problem.

12. Conclusions

In this paper we have presented an overview of a successful evolutionary approach to QC and RC synthesis. Even without heuristics we were able to find correct circuits representing universal gates, although perhaps non minimal ones. We find solutions to all quantum circuit synthesis problems presented by previous authors, such as entanglement and teleportation, and our programs reinvented several realizations of known gates such as Toffoli or Fredkin. In addition, we found very elegant solutions to gates such as Margolus, Miller and Peres. We believe that our solution to the Peres gate is minimal. It is now a challenge for future research to find a less expensive universal 3-qubit gate than the Peres gate using only 1-qubit and 2-qubit gates. The solution to this problem is not yet known, but in any case, based on our other studies, the Peres gate realization that we found seems to be very good, at least for NMR computing.

We hope that multiple circuit solutions will become eventually available from other authors for quantum circuit synthesis. We are compiling a library of "Benchmark Quantum Functions and their Circuits" [44] so that the future quantum software developers will be able to compare their results with those of our programs, analogously as it is a standard in current binary CAD.

In the course of this research we found the importance and mutual relations of four problems: circuit encoding, fitness function, cost function and local optimizing transformations. It was thanks to running the program and analyzing its various solutions that we found the sets of equivalence transformations and the heuristics for their application [55,57]. Although most examples were for 3-qubit circuits, our program allows in principle larger circuits. The RL variant is more efficient than the QL one.

Acknowledgments

The authors would like to thank Professor Soonchil Lee from Physics Department of KAIST in Korea for discussions about NMR computing and to Dr. Andrzej Buller from ATR Japan for his critical remarks on the first draft of this paper.

LITERATURE

1. M. Lukac, M. Pivtoraiko, A. Mishchenko, and M. Perkowski, "Automated Synthesis of Generalized Reversible Cascades using Genetic Algorithms", *Proc. Fifth Intern. Workshop on Boolean Problems*, pp. 33-45, September 19-20 2002, Freiberg, Sachsen, Germany.
2. M. Lukac, and M. Perkowski, "Evolving Quantum Circuits Using Genetic Algorithms", *Proc. of 5th NASA/DOD Workshop on Evolvable Hardware* 2002, pp. 177- 185.
3. 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.
4. M.D. Price, S.S. Somaroo, C.H. Tseng, J.C. Core, A.H. Fahmy, T.F. Havel and D.Cory, "Construction and Implementation of NMR Quantum Logic Gates for Two Spin Systems," *Journal of Magnetic Resonance*, 140, pp. 371-378, 1999
5. B.I.P. Rubinstein, "Evolving quantum circuits using genetic programming", *Proceedings of the 2001 Congress on Evolutionary Computation (CEC2001),* pp. 144-151, 2001.
6. 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.
7. C.P. Williams, and S.H. Clearwater, "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. D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, *Addison Wesley*, 1989.
10. A. Al-Rabadi, L. Casperson, M. Perkowski, and X. Song, "Canonical representation for Two-Valued Quantum Computing", *Proc. Fifth Intern.*

Workshop on Boolean Problems, pp. 23-32, September 19-20 2002, Freiberg, Sachsen, Germany.

11. R. Feynman, "Feynman Lectures on Computation", *Addison Wesley*, 1996
12. P. A. M Dirac, "The Principles of Quantum Mechanics", first edition, *Oxford University Press*, 1930.
13. J. von Neumann, "Mathematical Foundations of Quantum Mechanics", *Princeton University Press*, 1950.
14. E. Fredkin, and T. Toffoli, "Conservative Logic", *Int. J. of Theoretical Physics,* #21, 1982, pp. 219-253.
15. C. Bennett, "Logically Reversible Computation", *I.B.M. J. Res. Dev.,* #17, 1973, pp. 525-632.
16. A. Einstein, B. Podolsky, N. Rosen, *Physical Review* #47,1935, p. 777.
17. M. Klay, "Einstein-Podolsky-Rosen Experiments: The Structure of the Sample Space I, II", *Foundations of Physics Letters 1*, 1988, pp. 205-232
18. A. Graham, "Kronecker Products and Matrix Calculus with Applications", *Ellis Horwood Limited,* Chichester, UK, 1981
19. J. A. Wheeler, W. H. Zurek, "Quantum Theory and Measurement", *Princeton University Press*, 1983.
20. P. Kerntopf, "A Comparison of Logical Efficiency of Reversible and Conventional Gates", *Proc. of 3rd Logic Design and Learning Symposium (LDL),* Portland, Oregon, 2000.
21. J. Smolin, D. P. DiVincenzo, "Five two-qubit gates are sufficient to implement the quantum Fredkin gate." *Physical Review A*, Vol. 53, no. 4, April 1996, pp. 2855-2856.
22. Miller, D. M., "Spectral and Two-Place Decomposition Techniques in Reversible Logic," *Proc. Midwest Symposium on Circuits and Systems,* on CD-ROM, August 2002.
23. D. M. Miller and G.W. Dueck, " Spectral Techniques for Reversible Logic Synthesis," *Proc. RM 2003*, pp. 56-62.
24. G.W. Dueck and D. Maslov, "Garbage in Reversible Designs of Multiple-Output Functions," *Proc. RM 2003*, pp. 162 – 170.
25. K. Iwama, Y. Kambayashi, and S. Yamashita, "Transformation Rules for Designing CNOT-based Quantum Circuits," *Proc. DAC 2002,* New Orleans, Louisiana, pp. 419-424.
26. V.V. Shende, A.K. Prasad, I.L. Markov, J.P. Hayes, "Reversible Logic Circuit Synthesis," *Proc. 11th IEEE/ACM Intern. Workshop on Logic Synthesis (IWLS),* 2002, pp. 125 – 130.
27. A. Barenco et al., "Elementary Gates For Quantum Computation", *Physical Review A* 52, 1995, pp. 3457-3467
28. M.H.A. Khan, M. Perkowski, and P. Kerntopf, "Multi-Output Galois Field Sum of Products Synthesis with New Quantum Cascades," *Proceedings of 33rd International Symposium on Multiple-Valued Logic, ISMVL 2003*, 16-19 May 2003, Meiji University, Tokyo, Japan, pp. 146-153.
29. A. Al-Rabadi, "Novel Methods for Reversible Logic Synthesis and Their Application to Quantum Computing," *Ph. D. Thesis*, Portland State University, Portland, Oregon, USA, October 24, 2002.
30. M. Perkowski, P. Kerntopf, A. Buller, M. Chrzanowska-Jeske, A. Mishchenko, X. Song, A. Al-Rabadi, L. Jozwiak, A. Coppola, B. Massey, "Regular realization of symmetric functions using reversible logic,"

Proceedings of EUROMICRO Symposium on Digital Systems Design, 2001**,** pp. 245-252.

31. M. Perkowski, L. Jozwiak, P. Kerntopf, A. Mishchenko, A. Al-Rabadi, A. Coppola, A. Buller, X. Song, M. M. H. A. Khan, S. Yanushkevich, V. Shmerko, and M. Chrzanowska-Jeske, "A general decomposition for reversible logic," *Proceedings of RM 2001*. pp. 119 – 138.
32. A. Mishchenko and M. Perkowski, "Logic Synthesis of Reversible Wave Cascades", *Proc. IEEE/ACM International Workshop on Logic Synthesis,* June 2002, pp. 197 – 202
33. A. Khlopotine, M. Perkowski, and P. Kerntopf, "Reversible logic synthesis by gate composition," *Proceedings of IWLS* 2002, pp. 261 – 266.
34. P. Kerntopf, "Maximally efficient binary and multi-valued reversible gates," *Proceedings of ULSI Workshop*, Warsaw, Poland, May 2001**,** pp. 55-58.
35. A. Peres, "Reversible Logic and Quantum Computers," *Physical Review A*, 32:3266-3276, 1985.
36. J. Koza, "Genetic Programming. On the Programming of computers by means of natural selection," *The MIT Press,* 1992.
37. J. I. Cirac, P. Zoller, "Quantum Computation with Cold Trapped Ions," *Physical Review Letters* 74, Issue 20, 15 May 1995, pp. 4091-4094.
38. C. Monroe, D. Leibfried, B.E. King, D.M. Meekhof, W.M. Itano, and D.J. Wineland, "Simplified quantum logic with trapped ions," *Physical Review A* 55, Issue 4, April 1997, pp. 2489-2491.
39. C. Monroe, D.M. Meekhof, B.E. King, and D.J. Wineland, "A ''Schroedinger Cat'' Superposition State of an Atom," *Science* 272, May 24, 1996, pp. 1131-1136.
40. D.P. DiVincenzo, "Quantum Computation," *Science* 270, 1995, pp. 255-256.
41. A. Ekert, and R. Jozsa, "Quantum Computation and Shor's factoring algorithm," *Review of Modern Physics* 68, Issue 3, July 1996, pp. 733-753.
42. G. Negotevic, M. Perkowski, M. Lukac, and A. Buller, "Evolving quantum circuits and an FPGA based quantum computing emulator", *Proc. Fifth Intern. Workshop on Boolean Problems*, September 19-20 2002, Freiberg, Sachsen, Germany, pp. 15-22.
43. K. Dill, and M. Perkowski, "Baldwinian Learning utilizing Genetic and Heuristic for Logic Synthesis and Minimization of Incompletely specified data with Generalized Reed-Muller (AND-EXOR) forms", *Journal of System Architecture* 47, Issue 6, 2001, pp. 477-489.
44. www.ece.pdx.edu/~lukacm/q-bench.html
45. L.M.K. Van Der Sypen, M. Steffen, G. Breyta, C.S. Yannoni, M. H. Sherwood, and I.L. Chuang, "Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance**",** *Nature* 414, 883–887 (20/27 December 2001)
46. M. Hirvensalo, "Quantum Computing," *Springer Verlag*, 2001.
47. C. Vieri, M.J. Ammer, M. Frank, N. Margolus and T. Knight, A Fully Reversible Asymptotically Zero Energy Microprocessor*, MIT Artificial Intelligence Laboratory*, Cambridge, MA 02139, USA
48. M. A. Nielsen, and I.L. Chuang, "Quantum Computation and Quantum Information," *Cambridge University Press*, 2000.
49. J. Kim, J-S. Lee, and S. Lee, "Implementation of the refined Deutsch-Jozsa algorithm on a three-bit NMR quantum computer," *Physical Review A*, Volume 62, 022312, 2000

50. J. Kim, J-S. Lee and S. Lee, "Implementing unitary operators in quantum computation, *Physical Review A*, 032312, 2000.
51. J-S. Lee, Y. Chung, J. Kim, and S.Lee, "A Practical Method of Constructing Quantum Combinational Logic Circuits", *arXiv:quant-ph/9911053v1*,12 Nov. 1999.
52. M.D. Price, S.S. Somaroo, A.E. Dunlop, T.F. Havel, and D.G. Cory, "Generalized methods for the development of quantum logic gates for an NMR quantum information processor," *Physical Review A*, Vol. 60, No. 4, October 1999, pp. 2777-2780
53. Ch. Lomont, "Quantum Circuit Identities," *arXiv:quant-ph/0307111v1* 16 July 2003.
54. 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.
55. M. Lukac, S. Lee and M. Perkowski, "Inexpensive NMR realizations of quantum gates," *in preparation*, 2003.
56. G. Yang, W.N.N. Hung, X.Song, and M. Perkowski, "Majority-Based Reversible Logic Gate," *Proceedings of 6th International Symposium on Representations and Methodology of Future Computing Technology*, March 10-11, 2003, Trier, Germany, pp. 191-200.
57. M. Lukac, S. Lee and M. Perkowski, "Low cost NMR realizations of Ternary and Mixed Quantum Gates and Circuits," *in preparation*, 2003.
58. M. Perkowski, A. Al-Rabadi, and P. Kerntopf, "Multiple-Valued Quantum Logic Synthesis," *Proc. of 2002 International Symposium on New Paradigm VLSI Computing*, Sendai, Japan, December 12-14, 2002, pp. 41-47.