Techniques for the Synthesis of Reversible Toffoli Networks

D. MASLOV
University of Waterloo
G. W. DUECK
University of New Brunswick
and
D. M. MILLER
University of Victoria

We present certain new techniques for the synthesis of reversible networks of Toffoli gates, as well as improvements to previous methods. Gate count and technology oriented cost metrics are used. Two new synthesis procedures employing Reed-Muller spectra are introduced and shown to complement earlier synthesis approaches. The previously proposed template simplification method is enhanced through the introduction of a faster and more efficient template application algorithm, an updated classification of the templates, and the addition of new templates of sizes 7 and 9. A resynthesis approach is introduced wherein a sequence of gates is chosen from a network, and the reversible specification it realizes is resynthesized as an independent problem in hopes of reducing the network cost. Empirical results are presented to show that the methods are efficient in terms of the realization of reversible benchmark specifications.

Categories and Subject Descriptors: F. Theory of Computation, J.6 **[Computer-aided Engineering]**, B.6 **[Logic Design]** [**B.6.1 Design Styles, B.6.3 Design Aids**]: Computer-aided design (CAD), Automatic synthesis, Optimization, Switching theory

General Terms: Hardware, Theory of Computation

Additional Key Words and Phrases: quantum computing, reversible logic synthesis, circuit optimization

This work was supported by a Postdoctoral Fellowship (D. Maslov) and Discovery Grants (G. W. Dueck and D. M. Miller) from the National Sciences and Engineering Research Council of Canada.
Authors' addresses: D. Maslov: Institute for Quantum Computing and the Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON, N2L 3G1, Canada; email: dmitri.maslov@gmail.com; G. W. Dueck, Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B 5A3, Canada; email: gdueck@unb.ca; D. M. Miller, Department of Computer Science, University of Victoria, Victoria, BC, V8W 3P6, Canada; email: mmiller@uvic.ca

ACM Reference Format:

Maslov, D., Dueck, G. W., and Miller, D. M. 2007. Techniques for the synthesis of reversible Toffoli networks. ACM Trans. Des. Automat. Electron. Syst. 12, 4, Article 42 (September 2007), 28 pages. DOI = 10.1145/1278349.1278355 http:/doi.acm.org/10.1145/1278349.1278355

1. INTRODUCTION

The synthesis of reversible networks has received much attention in recent years (Agrawal and Jha 2004; Kerntopf 2004; Maslov and Dueck 2004; Maslov et al. 2005b; Miller 2002; Mishchenko and Perkowski 2002; Patel et al. 2004; Shende et al. 2003; Tsai and Kuo 2001). There are two primary motivations for this. One is power consumption. Landauer [1961] showed that irreversible circuits must consume power, and consequently dissipate heat, whenever they erase or otherwise discard information. Further, Bennett [1973] showed that for power not to be dissipated in an arbitrary circuit, it must be built from reversible gates. While the heat generation due to the information loss in modern CMOS (Complementary Metal Oxide Semiconductor) is still small, recent work by Zhirnov et al. [2003] shows the potentially prohibitive difficulty of heat removal with the increasing density of CMOS. The second motivator is that all quantum gates are reversible [Nielsen and Chuang 2000].

Hence there are compelling reasons to consider circuits composed of reversible gates and the synthesis of such networks. Advances in reversible logic synthesis can be applied to low-power CMOS design [Schrom 1998], quantum computing [Nielsen and Chuang 2000], and nanotechnology [Merkle 1993a; 1993b]. Quantum computing seems to be the most promising technology in terms of its potential practical use. Accordingly, our software provides the option of minimizing either the gate count or a quantum cost (in fact, any weighted linear cost function) of the resulting implementation. Research on reversible synthesis is of particular importance to the development of quantum circuit construction (in particular, oracles) and may well result in much more powerful computers and computations.

In this article, we develop a set of techniques for the reversible circuit synthesis and present a CAD (Computer-Aided Design) tool. There are many challenges for the designer of such a tool: the small size of modern quantum processors (the state of the art quantum processor can work with only 7 [Vandersypen et al. 2001] or 8 [Häffner et al. 2005] qubits; limited control has been recently demonstrated on a 12-qubit system [Negrevergne et al. 2006]), the difficulty in constructing a high fidelity implementation of the gates in existing hardware, and decoherence. Below are desirable CAD tool properties and an explanation of how they are addressed in this paper:

(1) *Reliability:* Our CAD tool is guaranteed to find a solution. It is important to have a network, even if it is not optimal.
(2) *Scalability:* Our software synthesizes circuits for functions with up to 21 variables in reasonable time. This allows synthesis of larger specifications than those that can be readily be implemented on the existing (quantum) hardware. The limiting factor is the size of the truth table that needs to

be stored in memory. In Section 9 we indicate how to improve the existing software to allow synthesis of larger specifications.

(3) *Quality:* Optimal or near-optimal networks are the goal of synthesis. Specific concerns for quantum technology include limited computational time due to decoherence and inaccuracy in applying the gates leading to accumulation of the errors, among a number of other issues. Thus, the size of a design is much more important when dealing with quantum technology than it is in other domains, such as CMOS. Results shown in Section 8 indicate that we have succeeded in this goal.

(4) *Runtime:* We are able to synthesize most benchmark functions within minutes. Some of our designs may take up to 12 hours to synthesize on an Athlon 2400XP machine with 512M of RAM memory running Windows. However, in Section 9 we discuss how to speed up our tool by a factor of 6 on a 6-processor parallel machine.

The remainder of the article is organized as follows. Section 2 provides the necessary background. In Section 3, we present an iterative synthesis approach that selects Toffoli gates so that the complexity of the Reed-Muller spectra specifying the reversible function is iteratively reduced until the specification becomes the identity. The complexity of the procedure depends on the number of nonzero coefficients in the spectra. This algorithm does not always find a solution, but it frequently finds better solutions than those found by earlier methods such as the one presented in Maslov et al. [2005]. We follow this section with the description of a second Reed-Muller spectra based synthesis algorithm (Section 4). A significant advantage of this algorithm is its guaranteed convergence, and a lesser quantum cost in the worst case scenario than the previously presented methods [Maslov et al. 2005, Shende et al. 2006]. Together the new Reed-Muller techniques and the earlier approach in Maslov et al. [2005] yield significantly improved results.

As presented in Maslov et al. [2005], once an initial network is found, it can often be simplified through the application of templates. In Section 6, we present an improved approach to templates, including classification of all templates with up to 7 gates and some useful templates with 9 gates. The template matching algorithm of Maslov et al. [2005b] is replaced with a more efficient one. Our new matching algorithm is better in the sense that, unlike the previous algorithm, under certain conditions it is guaranteed to find all possible network reductions that such a templates based tool can find. It also runs faster.

A resynthesis approach is presented in Section 7. This method relies on the fact that any sequence of gates in a reversible network realizes a reversible function. The method randomly (under some constraints) selects a sequence of gates from the network and then applies synthesis methods and the templates to the reversible function defined by that sequence. If the network found by resynthesis is smaller, it replaces the selected sequence in the original network. While our current approach to resynthesis is rather naive, it does significantly reduce the size of the network in many instances, particularly for some of the larger benchmark problems.

Empirical results are given in Section 8. We present the results of applying our methods to a number of benchmark functions. Our methods are also shown to produce an excellent overall average for the synthesis of all 3×3 reversible functions, only 0.16% above the optimum. The paper concludes with a short summary and suggestions for future research.

2. BACKGROUND

Definition 2.1. An n-input, n-output, totally specified Boolean function $(y_1, y_2, \ldots, y_n) = f(x_1, x_2, \ldots, x_n)$ is *reversible* if it is a bijection, that is; each input pattern is mapped to a unique output pattern.

Using methods such as the ones described in Maslov and Dueck [2004]; Miller [2002]; and Tsai and Kuo [2001], a (possibly incompletely specified) multiple-output Boolean function can be transformed into a reversible function. These methods are not particularly efficient, and it is an open problem to find better ways to perform such a transformation while minimizing the overhead due to the addition of "constant inputs" and "garbage outputs" [Gershenfeld and Chuang 1998]. In this article, we assume a reversible specification as the starting point.

Given a reversible specification, there are many ways [Agrawal and Jha 2004; Kerntopf 2004; Maslov et al 2005b; Mishchenko and Perkowski 2002; Tsai and Kuo 2001] of constructing a reversible network using multiple control Toffoli gates defined as follows:

Definition 2.2. For the domain variables $\{x_1, x_2, \ldots, x_n\}$ a *multiple control Toffoli gate* has the form $TOF(C;t)$, where $C = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$, $t = \{x_j\}$ and $C \cap t = \emptyset$. It maps the Boolean pattern $(x_1^0, x_2^0, \ldots, x_n^0)$ to $(x_1^0, x_2^0, \ldots, x_{j-1}^0, x_j^0 \oplus x_{i_1}^0 x_{i_2}^0 \ldots x_{i_k}^0, x_{j+1}^0, \ldots, x_n^0)$. The set C that controls the change of the j-th bit is called the set of *controls* and t is called the *target*.

Common special forms of this gate are: the NOT gate (a multiple control Toffoli gate with no controls) denoted $TOF(x_j)$, the CNOT gate (Controlled-NOT, a multiple control Toffoli gate with a single control bit) which is also known as a Feynman gate [Feynman 1985] and is denoted $TOF(x_i; x_j)$, and the original Toffoli gate (a multiple control Toffoli gate with two controls) denoted $TOF(x_{i_1}, x_{i_2}; x_j)$ [Toffoli 1980].

A reversible network is composed of reversible gates, which due to the restrictions dictated by the target technologies [Nielsen and Chuang 2000] form a cascade.

2.1 Cost of a Reversible Toffoli Network

It is a common practice in reversible logic synthesis research [Agrawal and Jha 2004; Kerntopf 2004; Maslov et al 2005b; Mishchenko and Perkowski 2002; Tsai and Kuo 2001] to synthesize a network using multiple control Toffoli gates and report its cost as a simple gate count. However, from the point of view of technological realization, multiple control Toffoli gates are not simple transformations. Rather they are composite gates themselves and Toffoli gates with a large set

of controls can be quite expensive [Barenco et al. 1995; Maslov et al. 2005]. We point out that there are three distinct Toffoli gate simulations [Barenco et al. 1995], one with an exponential cost and requiring no auxiliary bits, and two with linear costs, one requiring 1 and the other requiring $n-3$ auxiliary bits for an n-bit Toffoli gate. Given its exponential size and the usage of an infinite number of gate types requiring very accurate hardware realization due to the small rotation angles, we find multiple control Toffoli gate simulations with zero auxiliary bits impractical. Among the remaining two linear cost realizations of the Toffoli gates, the one associated with availability of $n-3$ auxiliary bits is smaller. Maslov et al. [2005] improves over the Toffoli gate simulation of Barenco et al. [1995] using the basis of elementary quantum gates NOT, CNOT, controlled-V and its inverse controlled-V^+ [Nielsen and Chuang 2000]. Such quantum gates were efficiently simulated in liquid state NMR (Nuclear Magnetic Resonance) quantum technology [Cory et al. 2000; Lee et al. 2006].

Definition 2.3. The *cost* of a Toffoli network $N = G_1G_2 \dots G_s$ with n inputs/outputs is a sum of costs of its gates, $Cost(N) := \sum_{i=1}^{s} Cost(G_i)$, which may sometimes be followed by an asterisk. The latter and costs of the individual gates are defined as follows.

—For a network with Toffoli gates of maximal size $n-1$, each k-bit ($k \leq n-1$) Toffoli gate cost is a minimum of the two linear cost realization gate counts [Maslov et al. 2005] as long as all associated auxiliary bits can be accommodated in the circuit. In this case, no asterisk will follow the numeric value of cost.

—For an n-bit network containing an n-bit Toffoli gate, during the calculation of the cost of each multiple control Toffoli gate we assume presence of an additional auxiliary bit (that is, assume that the network is built with $n+1$ lines; otherwise an exponential size Toffoli gate simulation must be employed [Shende et al. 2003], and we consider it to be inefficient). In such case, numeric value of the network cost is followed by an asterisk.

In this article, we report two sets of the synthesis results. In one, we minimize the gate count. This is done to compare the quality of our new approach to the quality of the previously presented methods. The second set of results contains networks synthesized to minimize the quantum cost defined above. The costs of the multiple control Toffoli gates are stored in a table. The software can easily accommodate different linear cost metrics.

2.2 Reed-Muller Spectra

Every Boolean function $y = f(x_1, x_2, \dots, x_n)$ can be uniquely written as a polynomial of the form $a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus a_3x_1x_2 \oplus \dots \oplus a_{2^n-1}x_1x_2 \dots x_n$ with Boolean coefficients $a_0, a_1, \dots, a_{2^n-1}$, which is referred to as the "positive polarity Reed-Muller expansion." A compact way to represent this expression is the vector ("spectrum") of coefficients $(a_0, a_1, \dots, a_{2^n-1})$. Given a size n reversible function, its Reed-Muller spectra (*RM spectra*) can be viewed as a table of size $n \times 2^n$, where each column represents the Reed-Muller spectrum of the corresponding output of the reversible function. Note that for reversible functions the last row

of this table is all zeroes and the size of the table can be reduced to $n \times (2^n - 1)$. RM spectra can be efficiently computed using fast transform techniques similar to a discrete Fast Fourier Transform (FFT). The transformation can be expressed in matrix form [Thornton et al. 2001] as

$$\begin{aligned} R &= M^n F \\ M^0 &= [1] \\ M^n &= \begin{bmatrix} M^{n-1} & 0 \\ M^{n-1} & M^{n-1} \end{bmatrix} \end{aligned}$$

where the summation is modulo-2, that is, EXOR, and F is the truth vector of the given function. In our software, this transformation, called RMT (Reed-Muller Transform), is implemented by the code shown below which maps a truth vector `f[ ]` of length 2^n given as an array of integers into the RM spectrum for the given function.

```
void RMT(int f[]){
  int i,j,k,m,p;
  int n = log(LengthOfVector(f[]));

  for (m=1;m<(2^n);m=2*m)
    for (i=0;i<2^n;i=i+2*m)
      for (j=i,p=k=i+m;j<p;j=j+1,k=k+1)
        f[k] = f[k] ^ f[j]; // bitwise EXOR
}
```

The elements of $f[\,]$ can be multibit values with each position representing a separate output function, in which case the procedure computes the RM spectra in parallel.

Important properties of this transformation include:

(1) *self inverse*, that is, $RMT\,(RMT\,(f)) = f$;
(2) *order dependence* in the sense that the value of $f[k]$ is never updated using any value of $f[j]$ where $j \geq k$;
(3) *power-of-two independence* in the sense that the value of $f[k]$ for $k = 2^s$ is never updated with the values of $f[j]$, where $j = 2^t$ and $1 \leq s, t \leq n$.

The RM spectra of the size n identity function with outputs $y_1, y_2, \ldots, y_n$ has a single nonzero coefficient $a_{2^{i-1}}$ for each y_i with all other coefficients 0.

Definition 2.4. The *RM cost* of a reversible function is the total number of coefficients for which its spectra differs from the spectra of the identity function.

We will refer to each nonzero row of the tabular representation of the RM spectra for the identity function as a *variable row*. Such variable rows are those at positions $1, 2, \ldots, 2^{n-1}$. We will also refer to all others as *nonvariable rows*.

2.3 Direct Application of a Toffoli Gate in RM Spectra

Application of a multiple control Toffoli gate $TOF(x_{i_1}, x_{i_2}, \ldots, x_{i_k}; x_j)$ from the input side of a reversible specification simply requires replacing each occurrence of the literal x_j in the Reed-Muller expansion of the output variable $y_s = a_0 \oplus a_{s,1}x_1 \oplus a_{s,2}x_2 \oplus a_{s,3}x_1x_2 \oplus \ldots \oplus a_{s,2^n-1}x_1x_2 \ldots x_n$ with the expression $x_j \oplus x_{i_1}x_{i_2} \ldots x_{i_k}$ followed by a simplification of the resulting expression. In the case where the Reed-Muller spectra is stored as a table this operation requires at most $n \times 2^n$ binary operations with no algebraic simplification. Application of a multiple control Toffoli gate $TOF(x_{i_1}, x_{i_2}, \ldots, x_{i_k}; x_j)$ from the output side can also be done directly in the spectra. In particular, the polynomials given by columns $y_{i_1}, y_{i_2}, \ldots, y_{i_k}$ (Boolean vectors of length 2^n) of the RM spectra are multiplied and the result is EXORed with column y_j with the result stored in column y_j. Hence, a Toffoli gate can be applied directly while working with the RM spectra. We note that since most reversible functions have numerous zero rows in their tabular RM spectra, it may be more efficient to store the indices and values of nonzero elements of the RM spectra. In this case, application of a Toffoli gate may require significantly fewer operations.

3. ITERATIVE NETWORK SYNTHESIS USING REED-MULLER SPECTRA

The first synthesis algorithm that we propose is very simple. At each step, by exhaustive enumeration it selects the Toffoli gate whose application to the function specification results in the largest decrease of the RM cost. If no gate application decreases the RM cost, a gate is chosen that results in the minimal increase of the RM cost. In both cases, if there is a tie between two or more gates, the gate with the smallest control set is chosen. If there is a tie based on the number of controls, our method selects the first gate in lexicographic order.

This synthesis approach is similar to some of earlier proposed techniques [Maslov and Dueck 2004; Miller 2002] in that the gates are chosen to decrease the function complexity (using a defined metric). We use a different gate library, and the Reed-Muller spectrum is used rather than the Walsh spectrum [Miller 2002] or Hamming distance defined over the truth table [Maslov and Dueck 2004], resulting in significantly better synthesis results. This is because the Reed-Muller spectrum corresponds more closely to the functional operation of Toffoli gates.

It is not surprising that there are drawbacks to such a simple approach. When considering larger benchmark specifications, we identified two major problems. First, the algorithm is not guaranteed to converge. In particular, it did not converge for the *hwb* type benchmark functions with more than 5 variables and function *ham*7. We address this problem by using the other algorithm (that always converges) first and taking its gate count as the upper bound for the synthesis using this algorithm. If the algorithm based on minimization of RM cost does not return a network with fewer gates than the second algorithm that we present below, its solution is considered inefficient. While such a technique appears efficient for the synthesis of benchmark functions, we are working on heuristics to force the Reed-Muller based method to converge on every reversible specification.

The second major problem with the new algorithm is that at each step it tries every possible Toffoli gate, of which there are $n \times 2^{n-1}$ for a size n reversible function. Current implementation of this algorithm uses a table to store the values of RM spectra, making the cost of the search for a single Toffoli gate assignment equal to $n^2 \times 4^{n-1}$ binary operations. In practice, it is too time-consuming to synthesize functions with more than 12 input variables (especially if the resynthesis technique discussed below is also used). We addressed this issue by having an option to limit the number of controls which every gate that we try might have. We plan to improve the runtime further by first exploring the idea noted above of storing only the nonzero coefficients; and second, searching for heuristics that can guide the selection of a Toffoli gate to avoid the current exhaustive enumeration. Ideas presented in Agrawal and Jha [2004] and Kerntopf [2004] might be useful. Further, we developed another synthesis algorithm that does not have these two major problems. This algorithm is outlined in the next section.

Despite the stated deficiencies, the iterative algorithm by itself converged for every one of the 40,320 different 3×3 reversible functions. It synthesized them with an average of 6.38 Toffoli gates per function in 3 seconds on a single 750 MHz processor Sun Blade 1000. This compares quite favorably to the 7.25 average for the MMD (Miller-Maslov-Dueck synthesis algorithm, [Maslov et al. 2005b] algorithm with no templates applied and shows that the iterative algorithm has very good potential.

4. MMD TYPE REED-MULLER SPECTRA BASED SYNTHESIS APPROACH

The second Reed-Muller based algorithm that we present is similar to MMD [Maslov et al. 2005b] in the sense that it works with a single row at a time, and allows a similar bidirectional modification. However, there are a number of differences between MMD [Maslov et al. 2005b] and the algorithm introduced in this section. Some of them are:

—our new algorithm works with Reed-Muller spectra, not in the Boolean domain (truth table) as does MMD;
—the choice of gates while working with a single row is completely different;
—at any point MMD does not change the correct form of upper rows, which is not true for the new method.

We start with a description of the unidirectional (basic) version of the algorithm. It consists of $2^n - 1$ steps (numbered 0 to $2^n - 2$). At each step i, the first i rows (rows with numbers $0, 1, \ldots, i-1$) in the tabular RM spectra of the function under synthesis match the first i rows of the RM spectra of the identity function. The algorithm assigns a (possibly empty) set of Toffoli gates such that the i^{th} row of the tabular RM spectra is transformed to match the i^{th} row of the RM spectra of the identity function. It can be observed that for such an algorithm, when step $2^n - 2$ is completed, the RM spectra is transformed to the RM spectra of the identity function, and thus the target specification is successfully synthesized. This is because row $2^n - 1$ of the RM spectra of a

reversible function is always zero. We now describe which gates are assigned depending on the row number and outline a proof showing that a suitable set of gates will always be found. We use $(r_n, r_{n-1}, \ldots, r_1)$ to denote the values in a row of the tabular representation for the RM spectra for the reversible function under consideration. We refer to a row as being *earlier* than another if it has a lower row index number.

A: Step $i = 0$. This step is unique since it is only for this step that we use NOT gates, and there is no need to consider if earlier rows are changed since there are none. Given the 0^{th} row has values $(r_n, r_{n-1}, \ldots, r_1)$ we apply NOT gates $TOF(x_j)$ for every $r_j = 1,\ j = 1..n$.

B: Step $i = 2^{k-1},\ k = 1..n$. Each of these rows is a variable row. Such a row, $(r_n, r_{n-1}, \ldots, r_1)$, has to be brought to the form $(0, 0, \ldots, 0, 1, 0, \ldots, 0)$ with 1 at a position k. This is done through the following two procedures. We first check if $r_k = 1$. If it is not, we make it equal one by assigning a gate $TOF\,(x_s; x_k)$ such that $s = \max\{j \mid r_j = 1,\ j = 1..n\}$ and $s > k$. According to Lemma 4.1 such an s exists, and it can be easily verified that application of the gate $TOF\,(x_s; x_k)$ does not affect RM spectra rows earlier in the table.

At this step the row we are working with has the form $(r_n, r_{n-1}, \ldots, r_{k+1}, 1, r_{k-1}, \ldots, r_1)$. We next use gates $TOF(x_k; x_j)$ for every $r_j = 1,\ j = 1..n$. By applying such gates we do not change rows earlier in the table than the row we are working with and at the same time the i^{th} row is transformed to the desired form $(0, 0, \ldots, 0, 1, 0, \ldots, 0)$ with 1 at position k.

C: Step $i,\ i \neq 2^k,\ i > 0$. For these i, we know that we are working with a non-variable row. Assume it has values $(r_n, r_{n-1}, \ldots, r_1)$. It has to be transformed to the form $(0, 0, \ldots, 0)$, which is the form of each non variable row of the RM spectra of the identity function. We first find $s = \max\{j \mid r_j = 1,\ j = 1..n\}$ such that item 2^{s-1} does not appear in the binary expansion of i. In other words, choose a variable whose i^{th} value in the RM spectra is 1 and that is not included in the product associated with the i^{th} element of the RM spectra. Such an s exists according to Lemma 4.1. We first apply gates $TOF\,(x_s; x_j)$ for every $r_j = 1,\ j \neq i,\ j = 1..n$. This transforms the row we are working with into $(0, 0, \ldots, 0, 1, 0, \ldots, 0)$ with 1 at position s. Second, we apply gate $TOF\,(X_i; x_s)$, where X_i is a product of variables x_j such that the j^{th} bit of the binary expansion of the number i equals one. Such an operation transforms the row we are working with into the desired $(0, 0, \ldots, 0)$. Finally, we undo $TOF\,(x_s; x_j)$ if such gates changed RM spectra rows earlier in the table than row i. Clearly, such "undo" operations do not change the correct form of the pattern we are working with.

To complete the proof of convergence for the above algorithm we need to show that at Steps B and C a value s with the required properties can always be found. The following lemma proves this.

LEMMA 4.1. *Suppose the RM spectra of a reversible function f has its first i rows (rows with numbers $0, 1, \ldots, i-1$) equal to the first i rows of the identity function. Denote the i^{th} row value by $(r_n, r_{n-1}, \ldots, r_1)$. Then,*

Table I. Synthesis of an Example Function Stored as a RM Spectra

	Function	Step1	Step2	Step3	Id
Spectral coef. of	cba	cba	cba	cba	cba
1	001	000	000	000	000
a	001	001	001	001	001
b	010	010	011	011	010
ab	000	000	000	000	000
c	100	100	100	100	100
ac	011	011	010	000	000
bc	011	011	010	000	000
abc	000	000	000	000	000
Gate applied:	$TOF(a)$ ↗	$TOF(b;a)$ ↗	$TOF(a,c;b)$ ↗	$TOF(b;a)$ ↗	

The result of application of the gate on the bottom of each column is shown in the following column while reading from left to right.

—*If* $i = 2^{k-1}$ $(k = 1..n)$, *then* $(r_n, r_{n-1}, \ldots, r_1) \neq (0, 0, \ldots, 0)$.

—*If* $i = 2^{k-1}$ $(k = 1..n)$ *and* $r_k = 0$, *then the number s defined as* $\max\{j \mid r_j = 1,\ j = 1..n\}$ *is greater than k*.

—*If* $i \neq 2^k$ *and* $(r_n, r_{n-1}, \ldots, r_1) \neq (0, 0, \ldots, 0)$, *then there exists s,* $1 \leq s \leq n$ *such that* 2^{s-1} *does not appear in the binary expansion of i and* $r_s = 1$.

PROOF. First, we prove by contradiction that if $i = 2^{k-1}$, then $(r_n, r_{n-1}, \ldots, r_1) \neq (0, 0, \ldots, 0)$. Suppose $(r_n, r_{n-1}, \ldots, r_1) = (0, 0, \ldots, 0)$ and apply RMT. The RMT will transform $(r_n, r_{n-1}, \ldots, r_1) = (0, 0, \ldots, 0)$ at position $i = 2^{k-1}$ into itself due to the properties 2 and 3 (order dependence and power-of-2 independence) of the RMT and the fact all nonvariable rows earlier than the i^{th} row are zero. According to the property 1 (self inverse) of RMT we are in the Boolean domain now, and we have two rows, the 0^{th} and i^{th}, both equal to $(0, 0, \ldots, 0)$. This is a contradiction since a reversible function cannot have two equal rows in its truth table representation.

Proof of the second statement is similarly shown by contradiction. Assume that such an s (which does exist as a result of the proof of the first statement) is less than k. In that case $(r_n, r_{n-1}, \ldots, r_1)$ can be interpreted as a binary expansion of a number $C < 2^{k-1}$ since its largest digit is at a position right of k. After applying RMT we move to the Boolean domain and find that pattern $(r_n, r_{n-1}, \ldots, r_1)$ did not change. At the same time, higher in the table, at position C, we will find a pattern equal to $(r_n, r_{n-1}, \ldots, r_1)$. This is the contradiction.

The proof of statement 3 is similar to the two proofs above. Assume such an s does not exist. Then, $(r_n, r_{n-1}, \ldots, r_1)$ may contain ones only at those positions where the binary expansion of $i = (i_n, i_{n-1}, \ldots, i_1)$ has ones. RMT transforms $(r_n, r_{n-1}, \ldots, r_1)$ into $(r_n \oplus i_n, r_{n-1} \oplus i_{n-1}, \ldots, r_1 \oplus i_1)$, a pattern that may have ones only at positions where the binary expansion of i has ones. An equal pattern may be found in the truth table at position $i - r$, where r is an integer with binary expansion $(r_n, r_{n-1}, \ldots, r_1)$. Again having two equal patterns in the truth table is a contradiction. □

Example 1. Consider the 3-variable reversible function specified by the permutation [1, 0, 3, 2, 5, 7, 4, 6] in Boolean domain. The spectra for this function are shown in tabular form in the column labeled Function in Table I. We want

to select Toffoli gates to transform this specification into that of the identity (Table I, column Id).

The first row of the function specification does not match the first row of the RM spectra of the identity function. This can be fixed by applying the NOT gate $TOF(a)$. Application of this NOT gate from the output side transforms the specification into the one shown in Table I, column Step1. The first 5 rows in specification Step1 match the first 5 rows of the RM spectra of the identity. We need to transform the sixth row from 011 to 000. First, we decrease the number of ones by applying $TOF(b;a)$. This leads to specification Step2. Note that the third row of Step2 has also changed, which means that it has to be updated later. Next, transform the sixth row of Step2 into the desired form 000 by applying $TOF(a,c;b)$. This results in specification Step3. Finally, apply CNOT gate $TOF(b;a)$, which leads to the identity specification and thus the network $(TOF(b;a)\,TOF(a,c;b)\,TOF(b;a)\,TOF(a))$ was constructed. The gates have been identified from the output to the input.

4.1 Bidirectional Method

The following lemma suggests how a bidirectional modification can be developed.

LEMMA 4.2. *Suppose the RM spectra of a reversible function f has its first i rows equal to the first i rows of the identity function. Then, so does the RM spectra of f^{-1}, the inverse of f.*

PROOF. This statement is, obviously, correct if one works with the truth table representation. In particular, if function f maps a pattern j into itself in the truth table, so will the inverse function. Due to the property 2 (order dependence) of the RMT, the same holds for all j, $0 \le j < i$. □

Assume the first $i-1$ positions in the RM spectra of f match the first $i-1$ positions of the RM spectra of the identity, then according to Lemma 4.2 so do the first $i-1$ positions of the RM spectra of f^{-1}. Hence, every assignment of gates that transforms the i^{th} row of f to match the i^{th} row of the identity without changing earlier rows (such gates are assigned from the output side of the cascade) will also transform the i^{th} row of f^{-1} to match the i^{th} row of the identity. Analogously, an assignment of gates that "fixes" the i^{th} row of f^{-1} will transform the i^{th} row of the RM spectra of f to that of the i^{th} row of the identity spectra. In latter case, the gates are assigned to the input side of the cascade being built. The question of which specification to work with, that of the function or its inverse, is equivalent to the question of which side of the network to assign the gates to, the input side or the output side. This is why we call this modification bidirectional.

Our approach chooses between using the function or its inverse based on the cost associated with fixing the i^{th} row of the corresponding RM spectra. By choosing a smaller cost associated with such a transformation we hope to synthesize an overall smaller network. In the case of a tie, we base our decision on the RM cost of the remaining specification—preference is given to a set of transformations that yield lower RM cost. We base this decision on the belief that, on average, functions with smaller RM cost are simpler to synthesize.

Finally, when these criteria do not resolve the choice, the gates are assigned to the output side (working with RM spectra of the function). Perhaps, there are better heuristics to be found for the decision of which side to work with, and it would be both interesting and beneficial to explore them.

THEOREM 4.3. *For any reversible function of size n the network synthesized by either of the two methods (unidirectional or bidirectional) contains*

(1) *In the multiple control Toffoli gates library: at most n NOT gates, at most* $2(n-1)(2^n-n-2)+n^2$ *CNOT gates, and at most* $\binom{n}{k}$ *Toffoli gates with k controls for each* $k \in [2..n-1]$.
(2) *In NCT (NOT-CNOT-Toffoli) [Shende et al. 2003] library: at most n NOT gates, at most* $2n2^n + o(n2^n)$ *CNOT gates, and at most* $3n2^n + o(n2^n)$ *Toffoli gates (assuming an additional auxiliary bit is available; otherwise the circuit may not be constructible [Shende et al. 2003]).*
(3) *In NCV (NOT-CNOT-controlled-sqrt-of-NOT)[Maslov et al. 2005] library: at most* $11n2^n + o(n2^n)$ *NOT, CNOT, controlled-V and controlled-*V^+ *gates (again, assuming an additional auxiliary bit is available; otherwise the circuit may not be constructible).*

PROOF. Proof of the first statement is based on an analysis of the basic synthesis algorithm described above. First, at most n NOT gates are used at step 0 (**A:**) of the synthesis algorithm, and none are used thereafter. At most n CNOT gates are required at each of the steps $i = 2^k$ (**B:**), totaling to n^2 CNOT gates. At each step $i,\ i \neq 2^k,\ i > 0$ (**C:**) at most $2(n-1)$ CNOT gates are required. The number of such steps is $2^n - n - 2$, giving a grand total of $2(n-1)(2^n-n-2)+n^2$ CNOT gates. Finally, on each step $i,\ i \neq 2^k,\ i > 0$ (**C:**), assuming $i = 2^{i_1} + 2^{i_2} + \cdots + 2^{i_k}$, at most one multiple control Toffoli gate with control set $\{x_{i_1}, x_{i_2}, \cdots, x_{i_k}\}$ is used. Calculating the number of such Toffoli gates with k controls gives $\binom{n}{k}$.

Calculation of the result in NCT library is based on multiple control Toffoli gate realizations from [Barenco et al. 1995]. In NCV library, the result is based on multiple control Toffoli gate realizations from [Maslov et al. 2005] and formulas

$$\sum_{k=0}^{n} k\binom{n}{k} = n2^{n-1}; \quad \sum_{k=0}^{n/2} k\binom{n}{k} = n2^{n-2} + o(n2^n). \qquad \square$$

Item 2 of the Theorem 4.3 shows a tighter upper bound (under the natural assumption that a CNOT gate is no more expensive than a larger Toffoli gate) for our synthesis algorithm as compared to the upper bound of n NOT gates, n^2 CNOT gates and $9n2^n + o(n2^n)$ Toffoli gates for the synthesis algorithm in Shende et al. [2003]. We also note another feature of this algorithm that might be useful for a more robust algorithm implementation. Linear reversible functions (considered in Patel et al. [2004]) will always be synthesized using NOT and CNOT (linear) gates only. While synthesizing linear functions, it is sufficient to store only the zeroth and all variable rows of its RM spectra. This allows synthesis of size 1000×1000 linear reversible functions while making a minimal change to the existing software.

The synthesis algorithm described in this section synthesized all size 3 reversible functions with the average of 6.98 Toffoli gates per function (no templates applied) in 7 seconds on (1.8 GHz) AMD Athlon XP2400+ processor. Again, this compares favorably to the 7.25 average for the MMD algorithm. For larger benchmark specifications this synthesis algorithm usually constructs smaller quantum cost networks compared to the MMD and iterative algorithms discussed in the previous section. Both phenomena should probably be explained by the tighter upper bound in Theorem 4.3 as compared to the upper bound in Maslov et al. [2005b].

5. COMPARISON OF THE NEW ALGORITHMS WITH MMD

In this section we compare performance of the newly presented synthesis methods to the performance of the MMD method [Maslov et al. 2005b]. Table II lists the name and size of a benchmark function tested and the number of gates and quantum cost (Definition 2.3) calculated when the synthesis methods MMD, iterative RM spectra based and MMD-type RM spectra based are applied. From this test we draw the following conclusions. The iterative RM spectra based method generally produces the smallest circuits for small specifications. However, when tested on larger functions it may diverge (Div.) or take a long time to complete, and thus does not apply (N/A). For larger specifications, the RM spectra based MMD type method takes the lead as far as quantum costs are concerned, and the application of the original MMD method results in the smallest gate count. In scope of this paper, a smaller quantum cost is more desirable than a smaller gate count, because quantum cost is a better indication of the technological cost of implementing a circuit.

6. TEMPLATES

In previous sections we discussed how to obtain a Toffoli circuit given a function specification. Since optimal synthesis is not feasible, we employed a number of heuristics. The result of heuristic search is, usually, a non-optimal circuit specification. Thus, optimization techniques can be applied to such circuits. In particular, we investigate a form of local optimization technique, called templates.

Templates are a generalization of rewriting rules. A *rewriting rule* is defined as a procedure that takes 2 equivalent circuits (circuits that compute the same function) and replaces one with the other. If the cost of the circuit to be replaced is greater than the cost of the replacement circuit this leads to a circuit cost reduction. Rewriting rules, other then templates, were proposed in Iwama et al. [2002] and Shende et al. [2006] for gate count reduction and in Lomont [2003] and Nielsen and Chuang [2000] for quantum circuits.

In many reversible logic synthesis papers, the cost of a network is defined as a weighted gate count. We refer to this as a *linear cost circuit metric*. In the more general case of a *nonlinear cost metric*, the cost of the complete circuit does not relate to the gates in a simple linear manner. An example of such a situation can arise when considering the Peres gate [Peres 1985] which, when simulated by a Toffoli gate (cost 5) and a CNOT gate (cost 1) would have a cost

Table II. Testing Performance of the Synthesis Methods

name	size	MMD GC	MMD QC	Iter. GC	Iter. QC	RM-based GC	RM-based QC
3_17	3	**6**	**14**	**6**	**14**	7	15
4_49	4	16	72*	**15**	**71***	20	72*
4mod5	5	9	25	**7**	**15**	9	25
5mod5	6	18	177*	**12**	**85***	18	177*
add3	4	6	18	**5**	**13**	6	14
cycle10_2	12	**19**	**1206**	27	1569	**19**	**1206**
cycle17_3	20	**48**	**6069**	N/A	N/A	**48**	**6069**
ham3	3	**6**	**10**	7	11	**6**	**10**
ham7	7	**25**	93	Div.	Div.	31	**57**
ham15	15	**138**	2145	N/A	N/A	159	**264**
hwb4	4	18	70*	**12**	**48***	16	56*
hwb5	5	57	481*	55	569*	**53**	**183**
hwb6	6	**134**	1723*	Div.	Div.	149	**816***
hwb7	7	**302**	5528*	Div.	Div.	435	**3036***
hwb8	8	**688**	15527*	Div.	Div.	1101	**7699***
hwb9	9	**1625**	48384*	Div.	Div.	2787	**22284***
hwb10	10	**3694**	124022*	Div.	Div.	6291	**49303***
hwb11	11	**8312**	343654*	Div.	Div.	14566	**126709***
mod5adder	6	37	591*	**24**	**242**	63	524*
mod1024adder	20	**55**	**1575**	N/A	N/A	**55**	**1575**
rd53	7	20	181	**19**	**113**	**19**	181

of 6, but when constructed directly in terms of quantum primitives has a cost of 4. When elementary quantum blocks are decomposed into pulses (as is done in liquid NMR quantum technology), similar nonlinear cost effects can arise.

We call a rewriting rule *regular* if the replacement circuit has smaller cost, otherwise we call it *irregular*. The qualifier regular is omitted when it is clear in context. The idea of applying regular rewriting rules to transform subcircuits of a given circuit is a powerful tool for circuit cost reduction. Application of irregular rules may be helpful in techniques such as simulated annealing. The simplification procedure consists of two parts. First, find as many regular rewriting rules as possible, and second, apply them to reduce the cost of a given circuit. Straightforward application of such an approach to quantum circuit cost reduction can be found in Lomont [2003] and was proposed in Iwama et al. [2002] for reversible networks composed of multiple control Toffoli gates. However, there are potential problems with this approach in its simplest form.

—The number of regular rewriting rules is very large even for small parameters. For instance, assuming Toffoli type gates have unit cost, the number of regular rewriting rules for reversible binary networks where $k = 3$ gates are replaced with $s = 2$ gates in a network with $n = 3$ input/output variables is 180. It can be easily shown that this number grows exponentially with respect to each of the parameters k, s, and n.

—Often, rewriting rules are redundant in the sense that a $G_1G_2G_3 \rightarrow G_4G_5$ rewriting rule can be a derivative of a $G_2G_3 \rightarrow G_5$ rewriting rule if $G_1 = G_4$. Further, it can be shown that even the number of nonredundant rules grows exponentially on n, and likely grows exponentially on k and s (keeping $s < k$).

—It is possible that commuting some gates in a cascade may permit the application of rewriting rules that decrease the cost. These gate interchanges are frequently possible and do not change the linear term cost of a network. It is possible to account for such a situation, but it will require introduction of a significant number of large rewriting rules. The templates matching algorithm we introduce takes care of this additional complexity associated with interchangeability of the gates.

The following observations are useful to understanding the template approach.

Observation 1. For any network $G_0G_1 \dots G_{m-1}$ realizing function f, the network $G_{m-1}^{-1}G_{m-2}^{-1} \dots G_0^{-1}$ is a valid network[1] for the function f^{-1}. This of course includes the case where the cascade of gates realizes the identity, in which case the inverse function is also the identity. We use Id to denote both the identity function and a network realizing the identity function, the meaning being clear from the context.

Observation 2. For any rewriting rule $G_1G_2 \dots G_k \rightarrow G_{k+1}G_{k+2} \dots G_{k+s}$, its gates satisfy the following:

$$G_1G_2 \dots G_kG_{k+s}^{-1}G_{k+s-1}^{-1} \dots G_{k+1}^{-1} = Id.$$

[1]Toffoli gates are self inverses: every gate $G = G^{-1}$. Thus, applying templates to networks of Toffoli gates will not require introducing any new gate types.

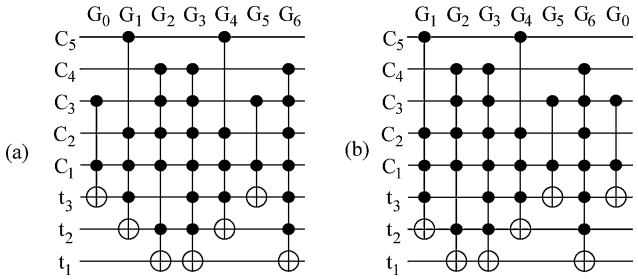

Fig. 1. Two cycles of a size 7 template: (a) can not be simplified using smaller and equal size templates and (b) can be simplified—gates G5, G6, and G0 can be replaced with two if template (5a) from Figure 3 is used.

Observation 3. For $G_0G_1 \dots G_{m-1} = Id$ and any parameter p, $0 \leq p \leq m$ $G_0G_1 \dots G_{p-1} \rightarrow G_{m-1}^{-1}G_{m-2}^{-1} \dots G_p^{-1}$ is a valid rewriting rule. In the most *trivial circuit cost metric*, where the cost of every gate is 1, *i.e.*, the gate count is calculated, the rewriting rule is regular for parameters p in the range $\frac{m}{2} < p \leq m$.

Observation 4. If $G_0G_1 \dots G_{m-1} = Id$, then $G_1 \dots G_{m-1}G_0 = Id$.

Observation 4 allows one to write an identity network with m gates in m (generally) different ways. We refer to each as a *cycle*. We are now ready to give the formal definition of a template.

Definition 6.1. A *size m template* is a cascade of m gates (a network) that realizes the identity function. For a cascade to be a template, there must be at least one cycle that can not be reduced in size (gate count) by the application of smaller or equally sized templates. Only the irreducible cycles are used when applying templates. A template $G_0\ G_1 \dots G_{m-1}$ can be applied in either direction:

(1) *Forward application* is a rewriting rule of the form

$$G_i G_{(i+1) \bmod m} \dots G_{(i+p-1) \bmod m} \rightarrow G_{(i-1) \bmod m}^{-1} G_{(i-2) \bmod m}^{-1} \dots G_{(i+p) \bmod m}^{-1},$$

where

$$0 \leq i, p \leq m-1.$$

(2) *Backward application* is a rewriting rule of the form

$$G_i^{-1} G_{(i-1) \bmod m}^{-1} \dots G_{(i-k+1) \bmod m}^{-1} \rightarrow G_{(i+1) \bmod m} G_{(i+2) \bmod m} \dots G_{(i-k) \bmod m},$$

where

$$0 \leq i, p \leq m-1.$$

Our earlier template definitions did not require the existence of a cycle that cannot be simplified, however, this part of the definition is important. We illustrate this with an example of a size 7 template with two cycles such that one simplifies and the other does not, shown in Figure 1. The network in Figure 1(a) does not simplify, whereas the one in Figure 1(b) can be simplified since its rightmost three gates can be replaced with two gates $TOF\ (t_2, t_3, C_1, C_2, C_3, C_4; t1)$ and $TOF\ (t_2, C_1, C_2, C_3, C_4; t_1)$.

Consistency of the template definition follows from the above four observations. One of the immediate benefits that the templates bring is significant reduction of the space required to store the rewriting rules (this is a significant improvement considering how much space is required in Shende et al. [2006] to store some small identities). In fact, one template occupies the same storage space as used by a single rewriting rule, yet it is capable of storing up to $2m^2$ non redundant rewriting rules. Assuming the trivial circuit cost metric where each gate has a cost of one, the number of regular nonredundant rewriting rules can be as high as m^2 for the odd numbers m and $m(m-1)$ for even m.

We earlier observed that the number of nonredundant rewriting rules grows exponentially; therefore, template classification is highly desirable. Depending on the set of model gates, classifications differ. We consider some of the particular questions and methods of proper classification of Toffoli templates in the next section.

6.1 Toffoli Templates

We wrote a program that helped us find the Toffoli templates. To build templates of size $s = s_1 + s_2$ the program first uses depth first search to find optimal networks of maximal sizes s_1 and s_2 using 3 to 4 input variables (which likely provides enough generality—we do not have a formal proof that it does—to find if a template is missing, but fails to generalize it once a candidate is found). In the second step, the program computes two sets with the truth vectors of functions realizable by cascades of sizes s_1 and s_2. Then, for every function in the first set it finds its inverse in the second set. If such a function is found the two networks are combined (use observations 1 and 2 to see that the resulting cascade is always the identity function) and templates of size less than s are applied to simplify the cascade. If this leads to a simplification for all cycles, the constructed identity is not a new template. Otherwise, it is a piece of a template and needs generalization.

The algorithm described finds those lines in a template that contain targets of the gates, but fails to extract all the possible assignments of the controls. Generalization requires finding all the possible gate controls that apply without changing the network functionality, that is, leaving it as the identity. The following theorem is useful as it limits the set of choices worth trying to assign the controls. The proof is straightforward and thus it is omitted. However, it is available directly from the authors upon request.

Definition 6.2. For any network $G_0G_1 \dots G_{m-1}$ with an input line that has controls only (control line), its *characteristic vector* $(\alpha_0, \alpha_1, \dots \alpha_{m-1})$, $\alpha_i \in \{0, 1\}$ for $0 \le i \le m-1$ has ones at positions i where the gate G_i has a control, and zeros everywhere else.

THEOREM 6.3.

(1) *If a control line with the characteristic vector* $(\alpha_0, \alpha_1, \dots \alpha_{m-1})$ *appears in a template of size m, any set of lines with this characteristic vector is a possible control set.*

(2) *Lines with characteristic vectors* $(0, 0, \dots 0)$ *and* $(1, 1, \dots 1)$ *are possible control lines for any template.*

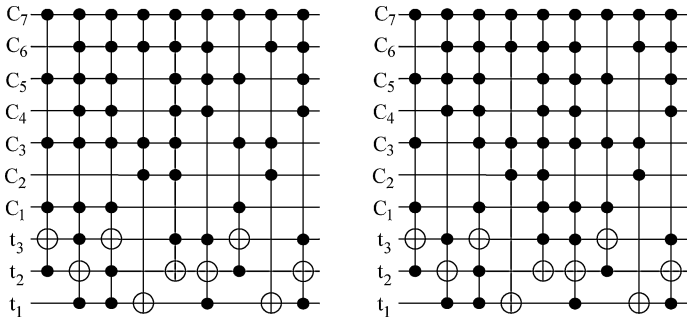

Fig. 2. Generalizations of a size 9 template.

(3) *If lines with characteristic vectors* $(\alpha_0, \alpha_1, \ldots \alpha_{m-1})$ *and* $(\beta_0, \beta_1, \ldots \beta_{m-1})$ *are control lines of a template, the line with characteristic vector* $(\alpha_0 \vee \beta_0, \alpha_1 \vee \beta_1, \ldots \alpha_{m-1} \vee \beta_{m-1})$ *is also a possible control line.*

(4) *If there exists a line with exactly two EXOR symbols on it, being targets of two gates* G_i *and* G_j*, every possible control line has* $\alpha_i = \alpha_j$.

Template generalization is a part of our software package. It is interesting to note that during the generalization process the number of templates may increase. Figure 2 illustrates how a template found by our program (bottom 4 lines: t_1, t_2, t_3 and C_1, $|C_1| = 1$) can be generalized in two different ways.

In Maslov et al. [2005b], we reported a Toffoli network with 4 gates for the 3-bit binary full adder. Assuming the trivial cost metric, we applied our templates. This resulted in no gate count reduction and we can conclude that the network is optimal for the given cost metric (gate count). Proof by contradiction. Suppose it is not. Then, there exists a smaller network for a 3-bit full adder, say with 3 gates. Using Observation 2 one can construct an identity cascade of size 7 built on 4 lines that would differ from the templates and will not be simplified by the means of the templates. Running our template finding program shows that it is impossible, and hence the network is optimal. The following theorem generalizes this observation.

THEOREM 6.4. *For the complete classification of the templates of size up to m and their complete (in the sense that no possible application is missed) application to network size reduction:*

—*For even numbers m, each subnetwork of size* $\frac{m}{2}$ *is optimal in any metric. The network itself is optimal if the number of gates is* $\frac{m}{2}$ *or less.*

—*For odd numbers m, each sub network of size* $\lfloor \frac{m}{2} \rfloor$ *is optimal in any metric, and each sub-network of size* $\lceil \frac{m}{2} \rceil$ *is optimal in the trivial metric. Similar statements hold for the entire network if the number of gates is not greater than* $\lfloor \frac{m}{2} \rfloor$ *or* $\lceil \frac{m}{2} \rceil$ *respectively.*

We conclude this section with a (most likely complete) list of the templates of size up to 7 and some templates of size 9, illustrated in Figure 3. Lines t_i in Figure 3 represent each a single line, and lines marked with C_j represent a possibly empty set of lines, all of the same form. We note that the templates

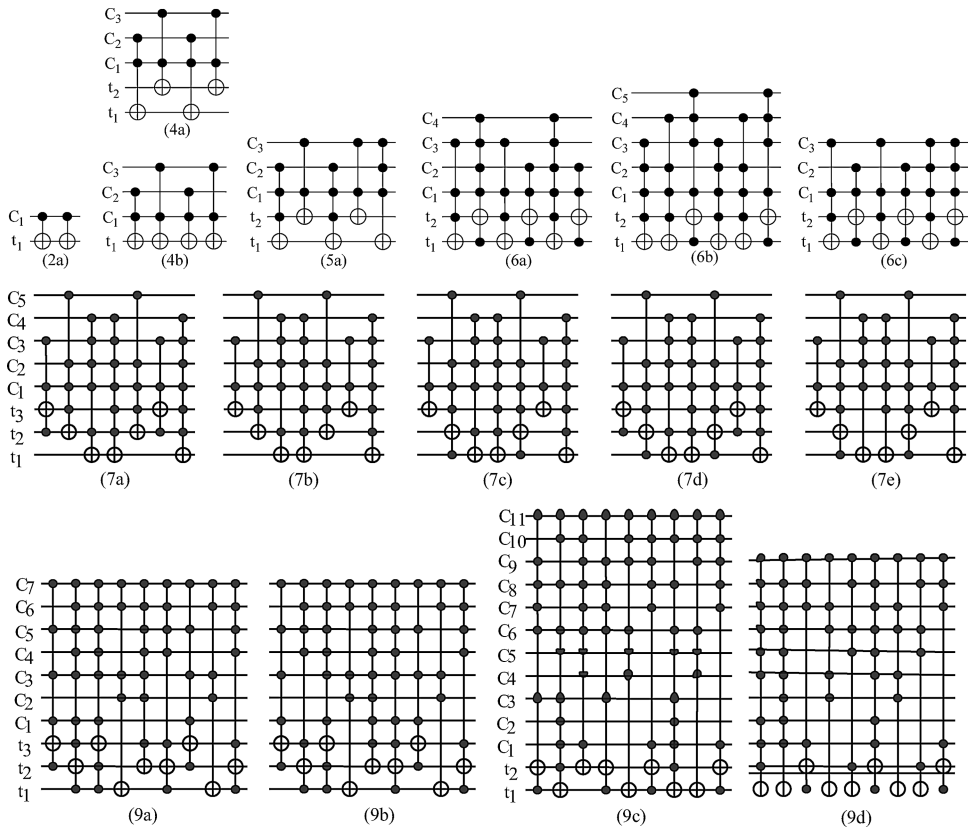

Fig. 3. All Toffoli templates of size up to 7, and some templates of size 9.

of size less than 5 are equivalent to those found in Maslov et al. [2005b]. We report a smaller number of different templates of size 6 compared to the templates reported in Maslov et al. [2005b]. There are two reasons for that. First, the template illustrated in Figure 8(d) of Maslov et al. [2005b] is not fully generalized, which we found with the help of our new software. And second, the template classification depends on how the templates are applied. Our algorithm for template application differs from the originally reported [Maslov et al. 2005b] and is discussed in the following section. A quick explanation of why the new algorithm is more accurate at finding more simplifications than the original is that template illustrated in Figure 8(b) of Maslov et al. [2005b] can now be simplified with the other templates which was not possible before. This is due to the improved matching algorithm.

6.2 Template Application

To apply templates to circuit cost reduction, we first consider all the templates of the form $ABAB$. Such templates applied for parameter $p = 2$ describe all gate commutations $AB \rightarrow BA$. That is, they define when the two adjacent gates in a cascade can be swapped. We call such templates *moving rules* and

apply them to move the gates in a cascade to permit the application of cost reducing template substitutions. The complete description of the templates of form $ABAB$ is very simple (this may of course not be true for all gate types). Assuming gate A has control(s) C_A and target T_A and gate B has control(s) C_B and target T_B, these two gates form a moving rule if, and only if, $T_A \notin C_B$ and $T_B \notin C_A$. It turns out that all Toffoli templates of size 4 are the moving rules.

Templates of size other than 4 are applied as follows. We choose a starting gate for matching. The position of the starting gate (*Start*) in the matching will change with time, and we begin with $Start = 2$. Suppose $Start = k$ in a cascade with n gates at the present time. We apply smaller templates first. They are easier to match, because one needs to find fewer gates to do the replacement, and in a sense smaller templates allow more general network transformation (for instance, applying size 2 templates can be described as deleting adjacent pairs of identical gates, while applying size 9 templates has no obvious simple description). For each of the templates, we match gate k in the network to the first gate in each of the m cycles of the template, which is always possible. We then try to find the gates in the network that match those in the template assuming the first gate of the template cycle matches this k^{th} gate in the network and trying both directions for the template application. Next we explain only how to apply a single template cycle in the forward direction, because application of other cycles and in the backward direction is analogous. At this point, we create two arrays, integer *MatchIndex*[ ] with one element k indicating that one gate at position k in the network is found and properly matched, and Boolean *MoveIndex*[ ] with one element equal to 1 and indicating that all gates can be moved to the one found (in this case no moving is required). In addition, integer $CurrentGate = k$ indicates that at the present moment we look at the gate k. To match more gates, we decrease *CurrentGate* by 1 and see if gate $k-1$ in the network matches the second gate in the template cycle. If it does, we increase the size of *MatchIndex* array by 1, and add $k-1$ to it. We also increase size of the *MoveIndex* array and add a new element, 1 to it. Since gate $k-1$ neighbors with gate k, there is no need to check if the gates can be moved together. Finally, we check if these 2 gates can be replaced with a smaller network using the present template cycle, and if they can, do the replacement and return $Start = k-1$. The template matching resumes with starting gate at position $k-1$ and by trying the smallest template first. If gate *CurrentGate* did not match the second gate in the template, we decrease integer *CurrentGate* by 1 (it is now equal to $k-2$) and see if this gate matches the second gate of the template cycle.

In general, if some s gates are matched and can be moved together (that is, $MatchIndex = [k_1, k_2, \ldots, k_s]$ and $MoveIndex = [m_1, m_2, \ldots, m_s]$ where *MoveIndex* contains a nonzero value indicating that the gates can be moved to the corresponding position), and a gate in a network at position $CurrentGate = k_{s+1}$ matches $(s+1)^{\text{st}}$ gate of the template cycle, the procedure for matching is as follows. First, we check if the gates can be moved together to each of the network positions $k_1, k_2, \ldots, k_s, k_{s+1}$. If the gates can be moved together, we update array *MoveIndex* to contain $s+1$ Boolean values that show where it is possible to move the gates. We next check if it is beneficial to replace the matched and movable together $s+1$ gates with the remaining $m-s-1$ of the given

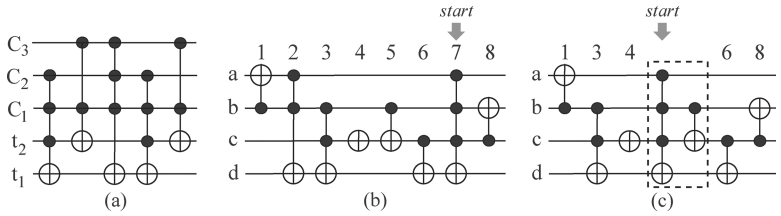

Fig. 4. Application of template (a) to the network (b) starting at the gate 7. The network simplifies to the form (c).

template. If it is, we do the replacement at the maximal value of an element of *MatchIndex*, k_j, corresponding to the nonzero value m_j of the *MoveIndex* and return new $Start = k_j - s + j$. The template matching resumes from this position in the network and the smallest template. If the gates cannot be moved together, we decrease value of the *CurrentGate* by one and try to match the gate at this position in the network. When *CurrentGate* becomes equal to zero or if we cannot match enough gates to do a beneficial replacement using the template, we try to match another cycle, the next template of the same size, or a larger template. If no templates match with a starting position *Start*, we increase its value by one (start matching with the next gate in the network) until we run out of gates in the network that could serve as a *Start*. In such a case, template application is completed.

We illustrate how the templates are applied with an example below.

Example 2. Consider network in Figure 4(b). Suppose $Start = 7$ and the template cycle that we want to match and apply is as illustrated in Figure 4(a). In other words, we start with the following known values *MatchIndex* = [7], *MoveIndex* = [1] and *CurrentGate* = 7. Line t_1 of the template must correspond to the line d of the network, and line t_2 should match one of a, b or c — this guarantees that gate 7 matches the first gate of the template cycle. The steps of matching are:

—Let *CurrentGate* $= 7 - 1 = 6$. Gate 6 does not match the second gate of the template cycle in Figure 4(a) since we expect to find a gate with target at a line where gate 7 has a control. As a result, *CurrentGate* gets decreased by 1.

—*CurrentGate* $= 5$. Gate 5 matches the second gate of the template if $t_2 = c$, $C_1 = \{b\}$ and $C_3 = \emptyset$. Gate 7 can be moved to gate 5, and gate 5 cannot move to the gate 7. Therefore, *MoveIndex* becomes [1, 0]. The *MatchIndex* is [7, 5]. We check that the replacement of the two gates we matched with the three reconstructed from the template is not beneficial, but since *MoveIndex* has non-zero values, we try to match more gates.

—*CurrentGate* $= 4$. Gate 4 in the network does not match the third gate in the template cycle because we are looking for a gate with the target on line d.

—*CurrentGate* $= 3$. Gate 3 in the network does not match third gate in the given template cycle because we try to find a gate with no control at line c.

—*CurrentGate* $= 2$. Gate 2 matches the third gate of the template cycle if $C_2 = \{a\}$. Gate 2 can be moved to gate 5, but gate 5 cannot be moved past

gate 3. Thus, $MoveIndex = [0, 1, 0]$. $MatchIndex = [7, 5, 2]$ and according to the template cycle, these three gates can be replaced with two. It is clearly beneficial to do the replacement. According to the *MoveIndex* the replacement can be done if all gates are moved to the gate 5 in the network. The network after template application is illustrated in Figure 4(c). Return $Start = 4$, because at this position the replacement part (in a dashed box) starts.

The template application resumes starting with the forth gate in the network in Figure 4(c) and trying to apply the smallest template.

In our program realization, function *apply_templates* is used to apply templates. It has an option of applying the templates to reduce the gate count or the quantum cost and works according to the algorithm discussed above. We made a modification of the matching algorithm in which we never look for the gates in the network further away from position *Start* than 20. This is because we found that in practice the gate span in template application is usually less than 20. Such restriction also makes the template algorithm faster—it is linear in the number of gates in the network. The template application algorithm from Maslov et al. [2005b] has an n^3 worst case scenario and n^2 best case scenario runtime in terms of the gate count of the circuit to be reduced. In addition, the template application algorithm introduced in this section reduces the circuits better (assuming both algorithms work with the same set of templates) than the one presented in Maslov et al. [2005b], and can be used in conjunction with different circuit cost metrics.

7. RESYNTHESIS PROCEDURE

In our program implementation, we first synthesize a function and its inverse using the MMD method [Maslov et al. 2005b] and the newly presented Reed-Muller spectra based algorithms. We then simplify each of the synthesized networks using templates, choose the smaller network N and declare it to be the final result. Each subnetwork N_{sub} of the final implementation is itself a network that computes some reversible function. This reversible function can be determined and synthesized on its own. We refer to such procedure as resynthesis. If such resynthesis yields a smaller subnetwork, it replaces N_{sub} leading to simplification of the overall network N.

We have implemented two drivers for this resynthesis procedure. First, a *random_driver* that performs a user-specified number of iterations. For each iteration, a number (again specified by the user) of random subnetworks are resynthesized and the best overall simplification is chosen and forwarded to the next iteration. Second is an *exhaustive_driver*. It tries all possible subnetworks with at least 5 gates of a given network. This is because it is not necessary to resynthesize networks with 4 or less gates, since in our synthesis approach every subnetwork of size 4 is optimal. This result is a corollary of Theorem 6.4.

The resynthesis procedure captures a somewhat similar idea for circuit optimization as peep-hole optimization introduced in Shende et al. [2006] for NOT-CNOT-Toffoli circuit simplification. Whereas [Shende et al. 2006] substitute a subcircuit with its optimal implementation, we synthesize the reversible

function a given subnetwork realizes in hopes of achieving a better implementation. The circuit we substitute is thus not guaranteed to be optimal. However, our resynthesis can be applied to subcircuits with a *large* number of variables (practically, 21), while optimal implementations can *only* be found and substituted for all 3-variable and some 4-variable circuits [Shende et al. 2006]. Since our work emphasizes synthesis of multiple control Toffoli gate networks, and the *majority* of non-empty subcircuits in benchmark function designs span over more than 4 bits, the simplification procedure from Shende et al. [2006] does not apply. The simplification procedure from Shende et al. [2006] can, obviously, be employed to synthesize optimal NOT-CNOT-Toffoli 3-bit reversible specifications. However, the focus of this paper is on the study of techniques applicable to the synthesis of larger specifications.

8. EMPIRICAL RESULTS

We have applied our synthesis approach to a number of reversible benchmark specifications from Maslov et al. [2005a][2]. We report two types of the results: gate count and quantum cost minimization. Gate count minimization is considered to make fair comparison to the previous approaches that used gate count as the target for minimization. We believe that circuits with minimized quantum costs are of a greater practical interest since quantum cost was defined as to reflect the expected cost of experimental implementation. The results can be found in Table III. The **name**, **size**, **GC** and **QC** columns give the name of each benchmark function, size (number of variables) of the reversible specification as considered in the literature, the best reported gate count, and the best reported quantum cost (as per Definition 2.3) for the networks with Toffoli gates. The next two columns report the gate count and the quantum cost when our tool is applied to synthesize a given function with the option of minimizing the gate count. The last two columns report the synthesis results with the option of quantum cost minimization. We find that realizations in the last two columns could be more practical. We note that networks for benchmark functions $4mod5$, $5mod5$, $hwb8 - hwb11$, and network for $rd53$ with quantum cost 79 found in Maslov et al. [2005a] are the results of the techniques discussed in this paper and were not reported before.

Table III shows our software synthesizes smaller[3] networks than earlier presented heuristics. For instance, the gate count for the $hwb6$ benchmark function was reduced from 126 to 42 gates, that is, our network is one third of the size of the best previously presented; and quantum cost for an implementation of this function was reduced by a factor of 10.

[2]In our comparison, we considered the networks and function specifications from the above web page. However, our quantum cost calculation differs from the one used in Maslov et al. [2005a], therefore quantum costs reported in Table III are slightly different (higher) from those that can be found online.

[3]Maslov et al. [2005a] contains networks synthesized using Toffoli and Fredkin gates, but we do not compare our results to those in a table form, just mention that the presented results are, generally, significantly better.

Table III. Benchmark Function Synthesis

Benchmark		Previous		Gate count minimization:		Quantum cost minimization:	
name	size	GC	QC	GC-gc	QC-gc	GC-qc	QC-qc
3_17	3	6	14	6	14	6	14
4_49	4	16	64*	12	32	12	32
4mod5	5	5	13	5	13	5	13
5mod5	6	10	85*	8	77*	10	71*
add3	4	4	12	4	12	4	12
cycle10_2	12	19	1206	19	1206	19	1206
cycle17_3	20	48	6069	48	6069	48	6069
cycle18_3	21	N/A	N/A	51	6819	51	6819
ham3	3	5	9	5	9	5	9
ham7	7	23	91	21	69	25	49
ham15	15	132	1881	70	463	109	214
hwb4	4	17	69*	11	23	11	23
hwb5	5	55	353*	24	114	24	114
hwb6	6	126	1519*	42	150	42	150
hwb7	7	289	5196*	236	3984*	331	2609*
hwb8	8	637	14636*	614	12745*	749	6197*
hwb9	9	1544	43138*	1541	43089*	1959	20378*
hwb10	10	3631	120034*	3595	117460*	4540	46597*
hwb11	11	9314	328200*	8214	336369*	11600	122144*
mod5adder	6	21	145	15	91	17	81
mod1024adder	20	55	1575	55	1575	55	1575
rd53	7	12	128	12	128	16	67
rd53	7	16	79	12	128	16	67

Gate count and quantum cost minimization column headers set the goal for our tool minimization criteria. Asterisks in the quantum cost calculation columns indicate necessity of the addition of a single qubit to the actual quantum implementation, refer to Definition 2.3. Actual circuits are available from Maslov et al. [2005a].

We limited the search time for our software to 12 hours for each benchmark function. Most functions took significantly less time to synthesize than the allowed 12 hours; most time (12 h) was spent to synthesize only one function, *cycle*18_3. A general rule was to synthesize a function using all three algorithms, apply the templates, resynthesize with *random_driver* until several iterations do not bring any simplification and apply *exhaustive_driver* until no further simplification occurs. In the chosen period of 12 hours, there was no time left to apply *exhaustive_driver* to functions (networks for) *hwb*7, *hwb*8, *hwb*9 and all networks with 10 and more variables other than *ham*15 and *cycle*10_2. Due to the time constraints, we did not apply *random_driver* to the networks for *hwb*11 and *cycle*18_3. Our software potentially can synthesize functions with more than 21 variables, but as the number of variables and gates in the synthesized network grows, the runtime for such synthesis grows exponentially.

In the literature, one of the common tests of the quality of a *heuristic* reversible synthesis method is how it performs on the 40,320 reversible functions with 3 inputs [Agrawal and Jha 2004; Kerntopf 2004; Maslov et al. 2005b]. We run this test and report the results only as a means of comparison of our technique to those heuristics reported earlier.

In our implementation of the discussed techniques, we used the 3 synthesis methods (MMD [Maslov et al. 2005b], iterative from Section 3, and bidirectional from Subsection 4.1) that are applied to both function and its inverse, then the templates were applied and the *exhaustive_driver* is run until no further simplification is found. This is a time-consuming test, and it takes around 96 hours for it to complete. Techniques to reduce the runtime are discussed in Section 9.

Table IV compares our synthesis results to the earlier reported synthesis algorithms and the optimal results found by depth-first search [Shende et al. 2003]. It can be seen that our results are significantly closer to the optimal synthesis than those reported earlier (by Maslov, Dueck, and Miller, column **MMD**; [Maslov et al. 2005b] and by Agrawal and Jha, column **AJ** [Agrawal and Jha 2004]). Our results are, on average (WA), only 0.16% off from the optimal size (column **Opt.** [Shende et al. 2003]). It can also be seen that our synthesis results are better than the best presented by Kerntopf [2004] (column **K**), even though that his work uses a larger gate library (given a larger gate library one would expect lower gate counts).

9. CONCLUSION

In this article, we presented new techniques for the synthesis of reversible Toffoli networks. The main contributions include two Reed-Muller spectra based approaches to reversible synthesis; a better characterization of templates and an improved method of their application, classification of the templates of size 7 (most likely complete) and demonstration of some useful templates of size 9. We also investigated an approach involving resynthesis of subnetworks that significantly improves the results, particularly for larger benchmark functions. We structured our software as to have an option of minimizing the gate count or a technology-motivated cost. To our knowledge, this is the first attempt to

Table IV. Number of reversible functions using a specified (column **Size**) number of gates for $n = 3$ for different synthesis algorithms as indicated by citations. The results we report in this paper are listed in column **Ours**. Column **K** is separated from the remainder of the table to stress that the gate library used in that work is different (larger). Row starting with WA reports the Weighted Average for the circuit gate count, and row % reports the same number but in percent with respect to the optimal size.

Size	MMD [Maslov et al. 2005b]	AJ [Agrawal and Jha 2004]	Ours	Opt. [Shende et al. 2003]	K [Kerntopf 2004]
13	6				
12	62				
11	391				
10	1444				
9	3837	30	2		86
8	7274	3297	659	577	2740
7	9965	12488	10367	10253	11774
6	9086	13620	16953	17049	13683
5	5448	7503	8819	8921	8068
4	2125	2642	2780	2780	3038
3	567	625	625	625	781
2	102	102	102	102	134
1	12	12	12	12	15
0	1	1	1	1	1
WA:	6.801	6.101	5.875	5.866	6.010
%	116%	104%	100.16%	100%	N/A

minimize a technology-motivated cost of the implementation in the relevant literature. We have implemented our methods in C++ and shown they produce results significantly better than those reported in the literature.

Further work can be done to optimize the code. For instance, our algorithm can be easily parallelized. Assuming a 6 processor machine, each of the 6 networks (3 methods, function and its inverse are synthesized) can be synthesized (including the template application) on a separate processor. The work of *random_driver* and *exhaustive_driver* can be distributed evenly among the processors. In total, such an algorithm on a parallel machine should be able to run almost 6 times faster as compared to a single processor machine. For large networks, template application can be parallelized by cutting them into smaller subnetworks and then applying the templates at the cutting points by restricting *Start* to grow no more than 20. The developed CAD tool should benefit from integrating into it the synthesis and circuit optimization techniques from the literature, such as those found in Agrawal and Jha [2004], Kerntopf [2004], and Shende et al. [2003, 2006].

Possible directions in the future research include identifying ways to synthesize reversible functions stored in compact formats (we suggest trying the associative table data format to store RM spectra), synthesizing incomplete specifications, as well as developing new heuristics and incorporating them (which, for the most part, should be straightforward) into the existing project.

ACKNOWLEDGMENTS

The authors wish to acknowledge the help of N. Scott, from the University of New Brunswick, who assisted in the preparation of this manuscript.

REFERENCES

AGRAWAL, A. AND JHA, N. K. 2004. Synthesis of reversible logic. In *Proceedings of the Conference and Exhibition on Design, Automation, and Test in Europe*. 21384–21385.

BARENCO, A., BENNETT, C. H., CLEVE, R., DIVINCENZO, D. P., MARGOLUS, N., SHOR, P., SLEATOR, T., SMOLIN, J. A., AND WEINFURTER, H. 1995. Elementary gates for quantum computation. *Phys. Rev. A*, *52,* 3457–3467.

BENNETT, C. H. 1973. Logical reversibility of computation. *IBM J. R&D*, *17*, 525–532.

CORY, D. G. , LAFLAMME, R., KNILL, E., VIOLA, L., HAVEL, T. F., BOULANT, N., BOUTIS, G., FORTUNATO, E., LLOYD, S., MARTINEZ, R., NEGREVERGNE, C., PRAVIA, M., SHARF, Y., TEKLEMARIAM, G., WEINSTEIN, Y. S., AND ZUREK, W. H. 2000. NMR-Based quantum information processing: achievements and prospects. Wiley Inter Science.

FEYNMAN, R. 1985. Quantum mechanical computers. *Optic. News*, *11*, 11–20.

GERSHENFELD, N. AND CHUANG, I. L. 1980. Quantum computing with molecules. *Scientific American*.

HÄFFNER, H., HÄNSEL, W., ROOS, C. F., BENHELM, J., CHEK-AL-KAR, D., CHWALLA, M., KÖRBER, T., RAPOL, U. D., RIEBE, M., SCHMIDT, P. O., BECHER, C., GÜHNE, O., DÜR, W., AND BLATT, R. 2005. Scalable multiparticle entanglement of trapped ions. *Nature*, *438*, 643–646.

IWAMA, K., KAMBAYASHI, Y., AND YAMASHITA, S. 2002. Transformation rules for designing CNOT-based quantum circuits. In *Proceedings of the IEEE/ACM Design Automation Conference*, 419–424.

KERNTOPF, P. 2004. A new heuristic algorithm for reversible logic synthesis. In *Proceedings of the IEEE/ACM Design Automation Conference*, 834–837.

LANDAUER, R. 1961. Irreversibility and heat generation in the computing process. *IBM J. R&D*, *5*, 183–191.

LEE, S., LEE, S., KIM, T., LEE, J., BIAMONTE, J., AND PERKOWSKI, M. 2006. The cost of quantum gate primitives. *J. Multiple-Valued Logic Soft Comp.*, *12*(5–6).

LOMONT, C. 2003. Quantum circuit identities. Arxiv quant-ph/0307111.

MASLOV, D. AND DUECK, G. W. 2004. Reversible cascades with minimal garbage. *IEEE Trans. Comput. Aid. Des.*, *23*, 11, 1497–1509.

MASLOV, D., DUECK, G., AND SCOTT, N. 2005a. Reversible logic synthesis benchmarks page. http://www.cs.uvic.ca/∼dmaslov/.

MASLOV, D., DUECK, G. W., AND MILLER, D. M. 2005b. Toffoli network synthesis with templates. *IEEE Trans. Comput. Aid. Des. 24*, 6, 807–817.

MASLOV, D., YOUNG, C., MILLER, D. M., AND DUECK, G. W. 2005. Quantum circuit simplification using templates. In *Proceedings of the Conference and Exhibition on Design, Automation, and Test in Europe*, 1208–1213.

MERKLE, R. C. 1993a. Reversible electronic logic using switches. *Nanotechn.*, *4*, 21–40.

MERKLE, R. C. 1993b. Two types of mechanical reversible logic. *Nanotechn.*, *4*, 114–131.

MILLER, D. M. 2002. Spectral and two-place decomposition techniques in reversible logic. In *Proceedings of the 45th IEEE Midwest Symposium on Circuits and Systems*. 493–496.

MISHCHENKO, A. AND PERKOWSKI, M. 2002. Logic synthesis of reversible wave cascades. In *Proceedings of the International Workshop on Logic and Synthesis*, pp. 197–202.

NEGREVERGNE, C., MAHESH, T. S., RYAN, C. A., DITTY, M., CYR-RACINE, F., POWER, W., BOULANT, N., HAVEL, T., CORY, D. G., LAFLAMME, R. 2006. Benchmarking quantum control methods on a 12-qubit system. *Physic. Rev. Lett.*.

NIELSEN, M. AND CHUANG, I. 2000 *Quantum Computation and Quantum Information*. Cambridge University Press.

PATEL, K. N., MARKOV, I. L., AND HAYES, J. P. 2004 Efficient Synthesis of Linear Reversible Circuits. In *Proceedings of the International Workshop on Logic and Synthesis*, 470–477.

PERES, A. 1985. Reversible logic and quantum computers. *Phys. Rev. A*, *32,* 3266–3276.

SCHROM, G. 1980 Ultra-low-power CMOS technology. PhD thesis, Technische Universität Wien, 1998.

SHENDE, V. V., PRASAD, A. K., MARKOV, I. L. AND HAYES, J. P. 2003. Synthesis of reversible logic circuits. *IEEE Trans. Comput. Aid. Des.* 226, 723–729.

SHENDE, V. V., PRASAD, A. K., PATEL, K. N., MARKOV, I. L., AND HAYES, J. P. 2006. Algorithms and data structures for simplifying reversible circuits. *ACM J. Emerg. Techn. Comp. Sys.*, 2(4).

THORNTON, M. A., DRECHSLER, R., AND MILLER, D. M. 2001. *Spectral Techniques in VLSI CAD*. Kluwer Academic Publishers.

TOFFOLI, T. 2001. Reversible computing. Tech memo MIT/LCS/TM-151, MIT Lab for Computer Science.

TSAI, I. M. AND KUO, S. Y., 2001. Quantum boolean circuit construction and layout under locality constraint. In *IEEE Conference on Nanotechnology*, 111–116.

VANDERSYPEN, L. M. K., STEFFEN, M., BREYTA, G., YANNONI, C. S., SHERWOOD, M. H., AND CHUANG, I. L. 2006. Experimental realization of Shor's quantum factoring algorithm using Nuclear Magnetic Resonance. *Nature*, *414*: 883–887.

ZHIRNOV, V. V., KAVIN, R. K., HUTCHBY, J. A., AND BOURIANOFF, G. I. 2003. Limits to binary logic switch scaling – a Gedanken model. *Proc. IEEE*, *91* 11, 1934–1939.

Received April 2006; revised November 2006, March 2007; accepted May 2007