Synthesis and Optimization of Reversible Circuits - A Survey

MEHDI SAEEDI, Amirkabir University of Technology
IGOR L. MARKOV, University of Michigan

Reversible logic circuits have been historically motivated by theoretical research in low-power electronics as well as practical improvement of bit-manipulation transforms in cryptography and computer graphics. Recently, reversible circuits have attracted interest as components of quantum algorithms, as well as in photonic and nano-computing technologies where some switching devices offer no signal gain. Research in generating reversible logic distinguishes between circuit synthesis, post-synthesis optimization, and technology mapping. In this survey, we review algorithmic paradigms — search-based, cycle-based, transformation-based, and BDD-based — as well as specific algorithms for reversible synthesis, both exact and heuristic. We conclude the survey by outlining key open challenges in synthesis of reversible and quantum logic, as well as most common misconceptions.

1. INTRODUCTION

A computation is *reversible* if it can be 'undone' in the sense that the output contains sufficient information to reconstruct the input, i.e., no input information is erased [Toffoli 1980]. It is also common to require that no information is duplicated. In Computer Science, reversible transformations have been popularized by the Rubik's cube and sliding-tile puzzles, which fueled the development of new algorithms, such as iterative-deepening A*-search [Korf 1999]. Prior to that, reversible computing was proposed to minimize energy loss due to the erasure and duplication of information. Today, reversible information processing draws motivation from several sources.

- **Considerations of power consumption** prompted research on reversible computation, historically. In 1949, John Von Neumann estimated the minimum possible energy dissipation per bit as $k_B T \ln 2$ where $k_B = 1.38065 \times 10^{-23}\mathrm{J/K}$ is the Boltzmann constant and T is the temperature of environment [Von Neumann 1966]. Subsequently, Landauer [1961] pointed out that the irreversible erasure of a bit of information consumes power and dissipates heat. While reversible designs avoid this aspect of power dissipation, most power consumed by modern circuits is unrelated to computation but is due to clock networks, power and ground networks, wires, repeaters, and memory. A recent trend in low-power electronics is to replace logic reversibility by charge recovery, e.g., through dual-rail encoding where the 01 combination represents a logical 0 and 10 represents a logical 1 [Kim et al. 2005].[1]
- **Signal processing, cryptography, and computer graphics** often require reversible transforms, where all of the information encoded in the input must be preserved in the output. A common example is swapping two values a and b without intermediate storage by using bitwise XOR operations $a = a \oplus b$, $b = a \oplus b$, $a = a \oplus b$. Given that reversible transformations appear in bottlenecks of commonly-used algorithms, new instructions have been added to the instruction sets of various microprocessors such as `vperm` in PowerPC AltiVec, `bshuffle` in Sun SPARC VIS, `permute` and `mix` in

[1] While charge recovery reminds *conservative logic* [Fredkin and Toffoli 1982], its essential property is to avoid dissipating electric charges by exchanging them. This property requires transistor-level support and is not specific to logic circuits as it also applies to clock networks.

M. Saeedi is currently with the Department of Electrical Engineering, University of Southern California.
Authors address: M. Saeedi, Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089-2562; email: msaeedi@usc.edu; I. L. Markov, Department of EECS, University of Michigan, Ann Arbor, MI 48109; email: imarkov@eecs.umich.edu.

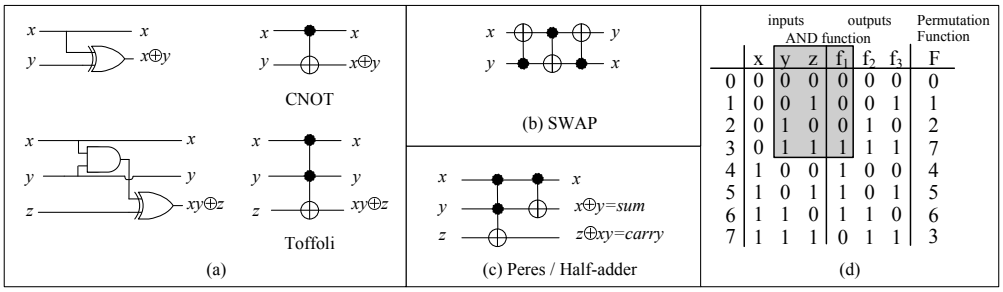
HP PA-RISC, `pshufb` in Intel IA32 and `mux` in Intel IA64 to improve their performance [McGregor and Lee 2003]. In particular, the performance of cryptographic algorithms DES, Twofish and Serpent, as well as string reversals and matrix transpositions, can be considerably improved by the addition of bit-permutation instructions [Shi and Lee 2000; Hilewitz and Lee 2008]. In another example, the reversible *butterfly* operation is a key element for Fast Fourier Transform (FFT) algorithms and has been used in application-specific Xtensa processors from Tensilica. Reversible computations in these applications are usually short and hand-optimized.

- **Program inversion and reversible debugging** generalize the 'undo' feature in integrated debugging environments and allow reconstructing sequences of decisions that lead to a particular outcome. Automatic program inversion [Glück and Kawabe 2005] and reversible programming languages [Yokoyama et al. 2008; De Vos 2010b] allow reversible execution. Reversible debugging [Visan et al. 2009] supports reverse expression watch-pointing to provide further examination of a problematic event.
- **Networks on chip** with mesh-based and hypercubic topologies [Dally and Towles 2003] perform *permutation routing* among nodes when each node can both send and receive messages. To route a message, regular permutation patterns such as bit-reversal, complement and transpose are applied to minimize the number of communication steps.
- **Nano- and photonic circuits** [Politi et al. 2009; Gao et al. 2010] are made up of devices without gain, and they cannot freely duplicate bits because that requires energy. They also tend to recycle available bits to conserve energy. Generally, building nano-size switching devices with gain is difficult because this requires an energy distribution network. Therefore, reversibility is fundamentally important to nano-scale computing, although specific constraints may vary for different technologies.
- **Quantum computation** [Nielsen and Chuang 2000] is another motivation to study reversible computation because unitary transformations in quantum mechanics are reversible. Quantum algorithms have been designed to solve several problems in polynomial time [Bacon and van Dam 2010; Childs and van Dam 2010], where best-known conventional algorithms take more than polynomial time.[2] A key example is number-factoring, which is relevant to cryptography. While unitary transformations can be difficult to work with in general, many prominent quantum algorithms contain large blocks with reversible circuits that do not invoke the full power of quantum computation, e.g., for arithmetic operations [Beckman et al. 1996; Van Meter and Itoh 2005; Takahashi and Kunihiro 2008; Markov and Saeedi 2012]. Circuits for quantum error-correction contain large sections of reversible circuits that implement GF(2)-linear transformations [Aaronson and Gottesman 2004].

In software and hardware applications of reversible information processing, sequences of reversible operations can be viewed as reversible circuits. For example, swapping two values x and y with a sequence of three XOR or CNOT gates (shown in Fig. 1a), operations $x = x \oplus y$, $y = x \oplus y$, and $x = x \oplus y$ is illustrated in Fig. 1b by a circuit. Such circuits are particularly useful in quantum computing. Reversibility prohibits loops and explicit fanouts in circuits,[3] and each gate must have an equal number of inputs and outputs with unique input-to-output assignments. Such pecu-

[2] BQP (Bounded-Error Quantum Polynomial-Time) is the class of problems solvable by a quantum algorithm in polynomial time with at most $\frac{1}{3}$ probability of error. P is the class of problems solvable by a deterministic Turing machine in polynomial time. Quantum computers have attracted attention as several BQP problems of practical interest are expected to be outside P.

[3] Read-only fanouts do not conflict with this requirement as illustrated by line x in Fig. 1c, and arbitrary fanouts can be simulated using ancilla lines, as we show in Fig. 3b.



	x	y	z	f_1	f_2	f_3	F
0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	1
2	0	1	0	0	1	0	2
3	0	1	1	1	1	1	7
4	1	0	0	1	0	0	4
5	1	0	1	1	0	1	5
6	1	1	0	1	1	0	6
7	1	1	1	0	1	1	3

Fig. 1. Expressing CNOT and Toffoli gates using AND and XOR gates (a), swapping two values x and y by three XOR operations (CNOT) as a reversible circuit (b), a reversible half-adder circuit (c), a sample reversible function (d).

liar features of reversible circuits prevent the use of existing algorithms and tools for circuit synthesis and optimization. *Reversible logic synthesis* is the process of generating a compact reversible circuit from a given specification. Research on reversible logic synthesis has attracted much attention after the discovery of powerful quantum algorithms in the mid 1990s [Nielsen and Chuang 2000]. Closely related techniques have also been motivated by other applications, e.g., the decomposition of permutations into tensor products is an important step in deriving fast algorithms and circuits for digital signal processing (Fourier and cosine transforms, etc.) [Egner et al. 1997].

This survey discusses methodologies, algorithms, benchmarks, tools, open problems and future trends related to the synthesis of combinational reversible circuits. The remaining part is organized as follows. In Section 2 basic concepts are introduced. We outline the process of reversible synthesis in Section 3, including optimization and technology mapping. Algorithmic details are examined in Sections 4 and 5. Available benchmarks and tools for reversible logic are introduced in Section 6. Finally, we discuss open challenges in reversible circuit synthesis in Section 7.

2. BASIC CONCEPTS

In this section, we introduce reversible logic gates and quantum gates, as well as reversible and quantum circuits. Representations of reversible functions and cost models for reversible gates are also discussed.

2.1. Reversible Gates and Circuits

Let A be a finite set and $f : A \to A$ a one-to-one and onto (bijective) function, i.e., a permutation. For instance, the function $g = (1, 5, 3, 2, 0, 4, 6, 7)$ is a permutation over $\{0, 1, \cdots, 7\}$ where $g(0) = 1$, $g(1) = 5$, $g(2) = 3$, etc. The set of all permutations on $A = \{0, 1, \cdots, 2^n - 1\}$ forms the *symmetric group*[4] S_n on A. A *reversible Boolean function* is a multi-output Boolean function with as many outputs as inputs, that is reversible. Fig. 1d illustrates a reversible function on three variables that implements the permutation $F = \{0, 1, 2, 7, 4, 5, 6, 3\}$.

Cycles. A *cycle* $(a_1, a_2, \cdots, a_k)$ is a permutation such that $f(a_1) = a_2$, $f(a_2) = a_3$, ..., and $f(a_k) = a_1$. For example, g can be written as $(0, 1, 5, 4)(2, 3)(6)(7)$. The *length* of a cycle is the number of elements it contains. A cycle of length two is called a *transposition*. A cycle of length k is called a *k-cycle*. 1-cycles, e.g., (6) and (7) in g, are usually

[4]In abstract algebra, a *group* is a set with a binary operation on it, viewed as multiplication, which is associative $(a \cdot b) \cdot c = a \cdot (b \cdot c)$, has a neutral element e such that $a \cdot e = e \cdot a = a$, and admits an inverse for every element $a \cdot a^{-1} = e$.

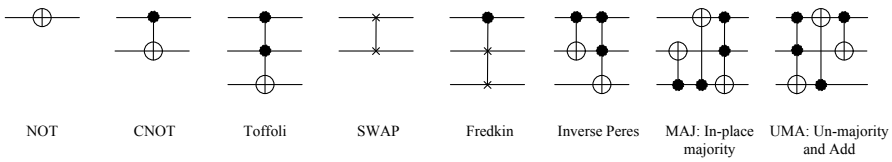


Fig. 2. Basic reversible gates. The Peres gate (reversible half-adder) is defined in Fig. 1c. The MAJ and UMA gates together form a full-adder gate, used in [Cuccaro et al. 2005] to build reversible multi-bit adders.

omitted. Cycles c_1 and c_2 are *disjoint* if they have no common members. Any permutation can be written as a product of disjoint cycles. This decomposition is unique up to the order of cycles. The composition of two disjoint cycles does not depend on the order in which the cycles are applied — disjoint cycles *commute*. In addition, a cycle may be written in different ways as a product of transpositions, e.g., $g = (0,1)(0,5)(0,4)(2,3)$ and $g = (4,5)(0,1)(1,5)(4,5)(0,4)(2,3)$. A cycle is *even* (*odd*) if it can be written as an even (odd) number of transpositions, i.e., a k-cycle is odd (even) if k is even (odd). The same definition applies to even and odd permutations in general.

Reversible gates. A *reversible gate* realizes a reversible function. For a gate g, the gate g^{-1} implements the inverse transformation. Common reversible gates are illustrated in Fig. 2.

- A *multiple-control Toffoli gate* [Toffoli 1980] C^{m}NOT$(x_1, x_2, \cdots, x_{m+1})$ passes the first m lines, *control* lines, unchanged. This gate flips the $(m+1)$-th line, *target* line, if and only if each positive (negative) control line carries the 1 (0) value. For $m = 0, 1, 2$ the gates are named NOT (N), CNOT (C), and Toffoli (T), respectively. These three gates compose the universal NCT library.
- A *multiple-control Fredkin gate* [Fredkin and Toffoli 1982] Fred$(x_1, x_2, \cdots, x_{m+2})$ has two target lines x_{m+1}, x_{m+2} and m control lines $x_1, x_2, \cdots, x_m$. The gate interchanges the values of the targets if the conjunction of all m positive (negative) controls evaluates to 1 (0). For $m = 0, 1$ the gates are called SWAP (S) and Fredkin (F), respectively.
- A *Peres gate* [Peres 1985] $P(x_1, x_2, x_3)$ has one control line x_1 and two target lines x_2 and x_3. It represents a C^{2}NOT(x_1, x_2, x_3) and a CNOT(x_1, x_2) in a cascade.
- An *in-place majority* (MAJ) gate computes the majority of three bits in place [Cuccaro et al. 2005], and provides the carry bit for addition. Cascading it with an *Un-majority and Add* (UMA) gate [Cuccaro et al. 2005] forms a full adder.

In multiple-control Toffoli and Fredkin gates, each line is either control or target. The order of controls is immaterial and so is the order of targets, but interchanging controls with targets will create a different gate. A multiple-control Toffoli (or Fredkin) gate implements a single transposition if only incident bit-lines are considered. The transposition is determined by the controls of the gate. If an extended set of bit-lines is considered, these gates will implement sets of disjoint transpositions.[5] Multiple-control Toffoli and Fredkin gates are self-inverse. For a Peres gate $P(x_1, x_2, x_3)$, the inverse Peres is the CNOT(x_1, x_2) C^{2}NOT(x_1, x_2, x_3) pair. For MAJ and UMA gates, the inverse gates can be constructed by reordering the CNOT and Toffoli gates.

Reversible circuits. A combinational *reversible circuit* is an acyclic combinational logic circuit in which all gates are reversible, and are interconnected without explicit fanouts and loops. In this survey, gates in a circuit diagram are processed from left to right. A reversible half-adder circuit in Fig. 1c implements the conventional half-

[5] If logic 0 and 1 are encoded as 01 and 10, respectively (dual-rail), SWAP performs inversion and the Fredkin gate models the function of the CNOT gate.

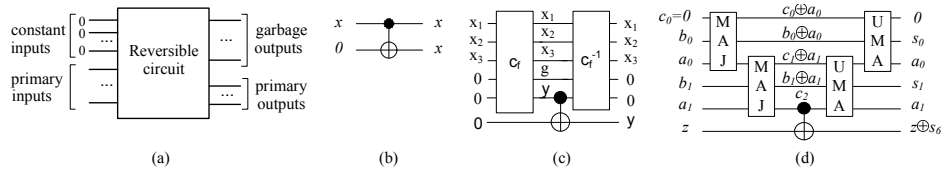


Fig. 3. The structure of inputs and outputs in a reversible circuit (a), explicit fanout in a reversible circuit with a CNOT gate and one ancilla (b), a reversible circuit for computing $y = f(x_1, x_2, x_3)$ described by C_f and garbage line g (c), a 2-bit ripple-carry adder [Cuccaro et al. 2005] (d).

adder when $z = 0$.[6] For a set of gates g_1, g_2, ..., g_k cascaded in a circuit C in sequence, the circuit $C^{-1} = g_k^{-1} g_{k-1}^{-1} \cdots g_1^{-1}$ (where g_i^{-1} is the inverse of g_i) implements the inverse transformation with respect to C. Different circuits computing the same function are considered *equivalent*. For example, circuits C_1=SWAP(x, y) and C_2 =CNOT(x, y) CNOT(y, x) CNOT(x, y) (Fig. 1b) are equivalent. For a library $\mathcal{L}$, an *$\mathcal{L}$-circuit* is composed only of gates from $\mathcal{L}$. A permutation is *$\mathcal{L}$-constructible* if it is computable by an $\mathcal{L}$-circuit. When the library consists of a single gate (type), we use the gate name instead of $\mathcal{L}$. We call permutations implementable with only NOT, CNOT, or Toffoli gates N-*constructible*, C-*constructible*, or T-*constructible*, respectively. S_n has 2^n N-constructible, $\prod_{i=0}^{n-1}(2^n - 2^i)$ C-constructible, and $(1/2)(2^n - n - 1)!$ T-constructible permutations [Shende et al. 2003]. Every even permutation is NCT-constructible [De Vos et al. 2002; Shende et al. 2003]. When dealing with n bits, reversible logic synthesis searches for solutions in a space of $O(n2^n)$ elements [Saeedi et al. 2010a]. A function f is *affine-linear*, or *linear* in short, if $f(x_1 \oplus x_2) = f(x_1) \oplus f(x_2)$ where $\oplus$ is a multi-bit XOR operation. NC-constructible permutations are linear functions and vice versa [Patel et al. 2008]. NCTSFP is the library consisting of NCT gates with SWAP, Fredkin and Peres gates added.

Ancilla lines. There are $2^n!$ distinct reversible functions on n variables which are permutations for 2^n elements. However, $\sum_{i=1}^{n} (2^i)^{2^n} \simeq 2^{n2^n}$ irreversible multiple-output (from 1 to n) functions exist. To make the specification reversible, input/output should be added. The added lines are called *ancillae* and typically start out with the 0 or 1 constant. An ancilla line whose value is not reset to a constant at the end of the computation is called a *garbage* line. Unconstrained outputs of ancillae lines in the truth table are called *don't cares* (DC). For an irreversible specification where each output combination can be repeated up to M times, $g = \lceil \log_2 M \rceil$ ancillae are required to build a reversible circuit [Maslov and Dueck 2004]. For example, at least two garbage lines (f_2 and f_3) and one constant line (x) are required to make the AND gate reversible as shown in Fig. 1d. Every odd permutation can be implemented with an NCT-circuit using one ancilla bit [Shende et al. 2003]. The Toffoli gate can be used with one constant line to compute the NAND function, i.e., C^{2}NOT$(a, b, 1)$, making Toffoli a universal gate in the Boolean domain. In general, the number of constant lines plus primary inputs is equal to the number of garbage lines plus primary outputs. See Fig. 3a for an illustration. A reversible copying gate or explicit fanout can be simulated by a CNOT and one ancilla, which leaves no garbage bit at the output as illustrated in Fig. 3b.

Reversible implementations. Toffoli [1980] proposed a generic NCT-circuit construction for an arbitrary reversible or irreversible function. For an implementation of

[6]Similar to the conventional arithmetic circuits that are typically designed in terms of half- and full-adders, identifying useful blocks such as half-adders is also common in reversible logic [Beckman et al. 1996; Van Meter and Itoh 2005; Cuccaro et al. 2005; Markov and Saeedi 2012].

any irreversible function $f(x)$, its reversible implementation can be described in the form $(x, y) \mapsto (x, y \oplus f(x))$. This specification is reversible since composing it with itself produces $(x, y \oplus f(x) \oplus f(x)) = (x, y)$. Given a conventional circuit for f, a reversible circuit can be constructed by making each gate reversible using a set of *temporary* lines if necessary. To reuse these temporary lines again, their values should be restored to their initial values. To restore the values, first copy function outputs to a set of ancillae with initial values 0 and then run the obtained reversible circuit in reverse to recover the starting values [Bennett 1973] as illustrated in Fig. 3c. Fig. 3d shows a 2-bit ripple-carry adder with one ancilla [Cuccaro et al. 2005] where values of a_0 and a_1 are recovered after computation. Note that if the values of temporary lines in a circuit are not restored, this circuit cannot be inverted and is not convenient as building blocks for larger circuits.

Representation models. Reversible functions can be described in several ways, as illustrated in Fig. 4.

- *Truth tables*. The simplest method to describe a reversible function of size n is a truth table with n columns and 2^n rows.
- *Matrix representations*. A Boolean reversible function (permutation) f can be described by a 0-1 matrix with a single 1 in each column and in each row (a *permutation matrix*), where the non-zero element in row i appears in column $f(i)$. A different matrix representation for linear functions [Patel et al. 2008] is described in Section 4.2.
- *Reed-Muller expansion*. To denote a specification with algebraic formula, *Positive polarity Reed-Muller* (PPRM) expansion can be applied. PPRM expansion uses only uncomplemented variables and can be derived from the EXOR-Sum-of-Products (ESOP) description by replacing a' with $a \oplus 1$ for a complemented variable a. The PPRM expansion of a function is canonical[7] and is defined as follows.

$$\begin{aligned} f(x_1, x_2, ..., x_n) = a_0 \oplus a_1 x_1 \oplus \cdots \oplus a_n x_n \oplus a_{12} x_1 x_2 \oplus \cdots \\ \oplus a_{n,n-1} x_{n-1} x_n \oplus \ldots \oplus a_{12\ldots n} x_1 x_2 \cdots x_n \end{aligned} \tag{1}$$

  A compact way to represent PPRM expansions is the vector of coefficients a_0, a_1, ..., $a_{12\ldots n}$, called the *RM spectrum* of the function. Consider an n-variable function and record its values (from the truth table) in a 2^n-element bitvector F. Then, the RM spectrum (R) of F over the two-element field[8] GF(2) is defined as $R = M^n F$ where

$$M^0 = [1], \;\; M^n = \begin{bmatrix} M^{n-1} & 0 \\ M^{n-1} & M^{n-1} \end{bmatrix} \tag{2}$$

- *Cycle expansion*. Viewing a reversible function as a permutation, one can represent it as a product of disjoint cycles.
- *Decision Diagrams*. A reversible function can be represented by a *Binary Decision Diagram* (BDD) [Bryant 1986; Hachtel and Somenzi 2000]. A BDD is a directed acyclic graph where the Shannon decomposition (i.e., $f = x_i f_{x_i=0} + x_i f_{x_i=1}$) is applied on each non-terminal node. Bryant [1986] proposed Reduced Ordered BDDs (ROBDDs), which offer canonical representations of Boolean functions. An ROBDD can be constructed from a BDD by ordering variables, merging equivalent sub-graphs and removing nodes with identical children. Several more specialized BDD variants have emerged for reversible and quantum circuits [Viamontes et al. 2009]. In general, a

[7]A canonical form is a way to rule out multiple representations of the same object. Given two different representations, they can be converted to canonical forms. The objects are equivalent if and only if the canonical forms match.

[8]The finite field GF(2) consists of elements 0 and 1 for which addition and multiplication are equivalent to logical XOR and AND operations, respectively.

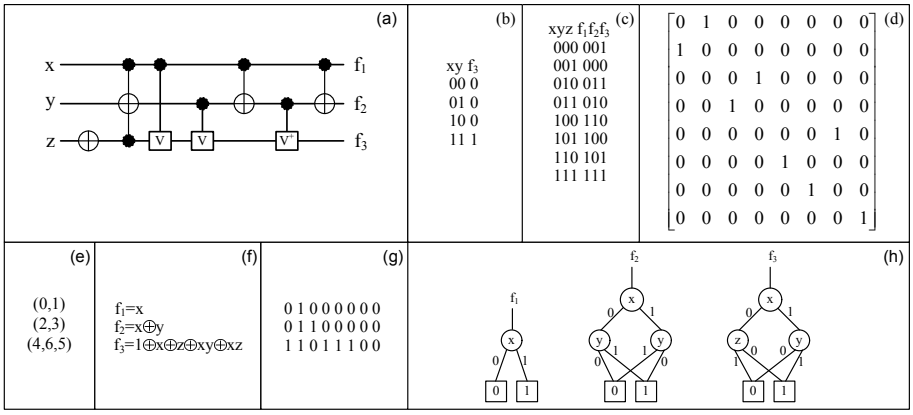


Fig. 4. A sample quantum circuit that implements a reversible specification (a), the specification in various formats: irreversible truth table (b), reversible truth table (c), matrix representation (d), cycle form (e), PPRM (f), RM spectrum (g), ROBDD (h).

BDD of a function may need an exponential number of nodes. However, BDD variants can represent many practical functions with only polynomial numbers of nodes.

Factorizations. A given cycle of length > 2 can be *factorized* into smaller cycles. For example, the 4-cycle $(0, 1, 5, 4)$ can be factorized into three 2-cycles $(0, 1)(0, 5)(0, 4)$. A factorization is of *type* $\alpha = (\alpha_2, \ldots, \alpha_k)$ if it results in exactly α_2 2-cycles, α_3 3-cycles and so on. Define $\langle\alpha\rangle = \Sigma_{j\geq 2}(j-1) \times \alpha_j$ for an n-bit permutation where α satisfies $\langle\alpha\rangle \geq n-1$. A factorization is *minimal* if $\langle\alpha\rangle = n-1$. For instance, the factorization $(0, 1, 5)(0, 4)(2, 3)$ of $g = (0, 1, 5, 4)(2, 3)$ is of type $\alpha = (2, 1)$ and is not minimal, $2 \times (2-1)+1 \times (3-1) = 4 > 2$. Two factorizations are *equivalent* if one can be obtained from the other by repeatedly exchanging adjacent factors that are disjoint. For example, factorizations $(0, 1, 5)(0, 4)(2, 3)$ and $(3, 2)(5, 0, 1)(0, 4)$ are equivalent. Since cycles $(0, 1, 5)$ and $(0, 4)$ share a common element, they do not commute.

2.2. Quantum Gates and Circuits

A quantum bit, *qubit*, can be treated as a mathematical object that represents a quantum state with two basic states $|0\rangle$ and $|1\rangle$. It can also carry a linear combination $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ of its basic states, called a *superposition*, where α and β are complex numbers and $|\alpha|^2$+$|\beta|^2$=1. Although a qubit can carry any norm-preserving linear combination of its basic states, when a qubit is *measured*, its state collapses into either $|0\rangle$ or $|1\rangle$ with probabilities $|\alpha|^2$ and $|\beta|^2$, respectively. A *quantum register* of size n is an ordered collection of n qubits. Apart from the measurements that are commonly delayed until the end of a quantum computation, all quantum computations are reversible.

Quantum gates. A matrix U is *unitary* if $UU^\dagger = I$ where $U^\dagger$ is the conjugate transpose of U and I is the identity matrix. An n-qubit *quantum gate* is a device which performs a $2^n \times 2^n$ unitary operation U on n qubits in a specific period of time. For a gate g with a unitary matrix U_g, its inverse gate g^{-1} implements the unitary matrix U_g^{-1}. Among various quantum gates with different functionalities [Nielsen and Chuang 2000] are Hadamard (H), phase shift (R_θ), controlled-V, controlled-V†, and the Pauli gates, which are defined in Fig. 5. For $\theta = \pi/2$ ($e^{i\theta} = i$) and $\theta = \pi/4$, the phase shift gate is named the Phase (P) and $\frac{\pi}{8}$ (T) gates, respectively. NCV is the library of NOT, CNOT, controlled-V and controlled-V†. For an arbitrary single-qubit gate U, a controlled-U gate is a 2-qubit gate with one control and one target which applies U

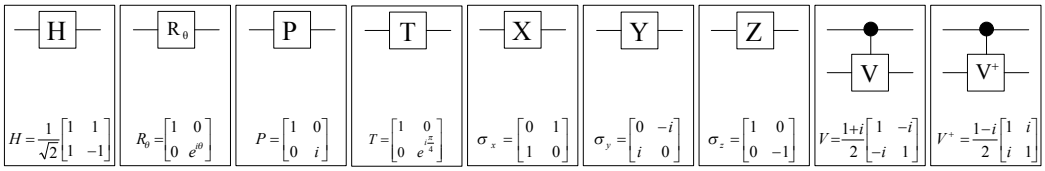


Fig. 5. Basic quantum gates: Hadamard, phase shift, Phase, $\frac{\pi}{8}$, Pauli-X (NOT), Pauli-Y, Pauli-Z, controlled-V, and controlled-V†.

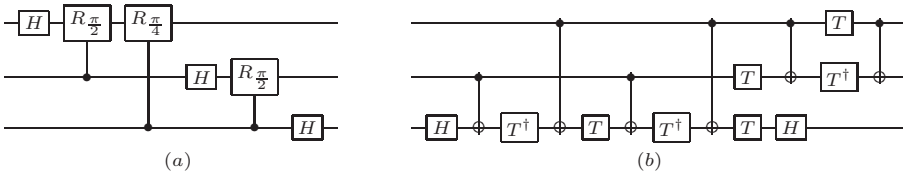


Fig. 6. A three-qubit Quantum Fourier Transform (a), decomposing the Toffoli gate into one-qubit and six CNOT gates; six CNOT gates are required [Shende and Markov 2009] (b).

on the target qubit whenever the control condition is satisfied. Basic quantum gates are illustrated in Fig. 5. The set of reversible gates is a subset of all possible quantum gates, distinguished by having only 0s and 1s as matrix elements. It would be misleading to call reversible circuits *quantum* just because they are used in quantum information processing. As we show in Section 2, reversible circuits can be described and manipulated without leaving the Boolean domain. The size of reversible circuits can sometimes be reduced by introducing non-Boolean gates (Section 5).

Quantum circuits. A *quantum circuit* consists of quantum gates, interconnected by qubit carriers (i.e., wires) without feedback and explicit fanouts. Fig. 6a illustrates a 3-qubit quantum circuit for the Quantum Fourier Transform (QFT) which includes the Hadamard and phase shift gates. The inverse of a quantum circuit is constructed by inverting each gate and reversing their order. A set of gates is universal for quantum computation if any unitary operation can be approximated with arbitrary accuracy by a quantum circuit which contains only those gates. The gate library consisting of CNOT and single-qubit gates is universal for quantum computation [Nielsen and Chuang 2000]. Fig. 6b shows a decomposition of the Toffoli gate into H, T, T†, and CNOT gates; six CNOTs are required for Toffoli [Shende and Markov 2009]. The search space for quantum-logic synthesis is not finite, and circuits implementing generic unitary matrices require $\Omega(4^n)$ gates [Shende et al. 2004].

Stabilizer circuits. The gates Hadamard, Phase, and CNOT are called stabilizer gates. A stabilizer circuit is a quantum circuit consisting of stabilizer gates and measurement operations. Stabilizer circuits have applications in quantum error correction, quantum dense coding, and quantum teleportation [Nielsen and Chuang 2000]. According to the Gottesman-Knill theorem [Nielsen and Chuang 2000], quantum circuits exclusively consisting of the following components can be efficiently simulated on a classical computer in polynomial time:

- A state preparation N-circuit with initial value $|000...0\rangle$ — qubit preparation in the computational basis,
- Quantum gates from the Clifford group (Hadamard, Phase, CNOT, and Pauli gates),
- Measurements in the computational basis

Evaluation and simulation of quantum circuits. For matrices $A_{m\times n}$ and $B_{p\times q}$, the tensor (Kronecker) product $A \otimes B$ is a matrix of size $mp \times nq$ in which each element of A is replaced by B. The unitary matrix effected by several gates acting on disjoint qubits (in parallel) can be calculated as the tensor (Kronecker) product of gate matrices. For a set of k gates g_1, g_2, ..., g_k with matrices U_1, U_2, ..., U_k cascaded in a quantum circuit C (sequentially), the matrix of C can be calculated as $U_k U_{k-1} ... U_1$. Straightforward simulation of quantum circuits by matrix multiplication requires $\Omega(2^n)$ time and space [Viamontes et al. 2009]. To improve runtime and memory usage, algorithmic techniques have been developed for high-performance simulation of quantum circuits [Shi et al. 2006; Viamontes et al. 2009].[9]

Quantum circuit technologies. To physically implement qubits, different quantum-mechanical systems have been proposed, each with particular strengths and weaknesses, as discussed in the *Quantum Computation Roadmap* (http://qist.lanl.gov/qcomp_map.shtml). Leading candidate technologies represent the state of a qubit using

- A two-level motion mode of a trapped ion or atom,
- Nuclear spin polarizations in nuclear magnetic resonance (NMR),
- Single electrons contained in Gallium arsenide (GaAs) quantum dots,
- The excitation states of Josephson junctions in superconducting circuits,
- The horizontal and vertical polarization states of single photons.

Quantum gates are effected by shining laser pulses on neighboring ions or atoms, applying electromagnetic pulses to spins in a strong magnetic field, changing voltages and/or current in a superconducting circuit, or passing photons through optical media. These and other technologies are discussed in textbooks [Nielsen and Chuang 2000] and research publications [Politi et al. 2009; Gao et al. 2010].

Interpreting quantum circuit diagrams. Representing quantum circuits with circuit diagrams invites analogies with conventional CMOS circuits, but there are several fundamental differences.

(1) In a quantum circuit, qubits typically exist as fixed physical entities (e.g., electrons, photons or nuclei), and quantum gates operate on a qubit register (some gates can be invoked in parallel). This is in contrast to conventional semiconductor circuits where signals travel through gates, often fanning out and reconverging.
(2) Wirelines in a quantum circuit are used to trace the different states of a qubit during computation. Unlike in conventional circuits, wirelines in quantum and reversible circuits have sequential semantics. This can be illustrated by considering constant-propagation, i.e., simplifying a circuit when some of the inputs are given known values. Even when the input values are 0 or 1, wirelines in circuits like the reversible adder from [Cuccaro et al. 2005] (Fig. 3d) cannot be removed because they are also used to store intermediate values of computation.
(3) In many implementations, quantum gates are invoked by electromagnetic pulses, in which case the different gates of a combinational circuit appear for short periods of time and then disappear. This is in contrast to more familiar circuits in semiconductor chips, where independently existing gates are connected by metallic interconnect. Photonic quantum circuits use explicit interconnect in the form of photonic waveguides.

[9] PP (Probabilistic Polynomial-Time) is the class of decision problems solvable by an NP (Nondeterministic Polynomial-Time) machine which gives the correct answer (i.e., 'Yes' or 'No') with probability $> \frac{1}{2}$. P^{PP} (P with PP oracle) includes decision problems solvable in polynomial time with the help of an oracle for solving problems from PP. Quantum circuit simulation belongs to the complexity class P^{PP}.

(4) Conventional circuits are typically synchronized through sequential elements (latches and flip-flops) because the timing of individual gates cannot be controlled accurately. In quantum circuits where each gate can be invoked at a precisely specified moment in time, there is no need for synchronization using sequential gates, and the entire computation can be scheduled by timing each combinational gate.
(5) In conventional circuits, each wire is assumed to carry a 0 or 1 signal, and each output of a combinational circuit is deterministically observable at the end of a clock cycle. However, these assumptions break down in a quantum circuit that generates non-Boolean values [Nielsen and Chuang 2000] because (i) multiple qubits can be entangled, (ii) to directly observe a qubit, it must be measured, which generates a nondeterministic outcome and affects other entangled qubits.

These differences between quantum and conventional circuits are sometimes misunderstood in the literature, as we point out in Section 7.

2.3. Circuit Cost Models

Current quantum technologies suffer from intrinsic limitations which prohibit some circuits and favor others, prime examples are the small number of available qubits and the requirement that gates act only on geometrically adjacent qubits (in a particular layout). To be relevant in practice, circuit synthesis algorithms must be able to satisfy technology-specific constraints and improve technology-specific cost metrics. For example, currently popular trapped-ions [Häffner et al. 2005] and liquid-state NMR [Negrevergne et al. 2006] technologies allow computation on sets of 8-12 qubits in a linear nearest neighbor (LNN) architecture where only adjacent qubits can interact. Furthermore, a physical qubit can hold its state only for a limited time, called *coherence time*, which varies among different technologies from a few nanoseconds to several seconds [Van Meter and Oskin 2006]. Because of decoherence, qubits are fragile and may spontaneously change their joint states.

Just as in conventional circuits, the trivial *gate count* metric does not adequately reflect the resources required by different gates. Similar to transistor counts, used to compare logic gates implemented in CMOS chips, one can define the technology-specific cost of quantum gates by decomposing them into elementary blocks supported by a particular technology. A physical implementation of an elementary operation depends on the Hamiltonian[10] of a given quantum system [Zhang et al. 2003]. For example in a one-dimensional exchange, i.e., Ising Hamiltonian characterized by interaction in the z direction only, the 2-qubit SWAP gate requires three qubit interactions. In a two-dimensional exchange with the XY Hamiltonian, it can be implemented by a single two-qubit interaction. In an ion trap system, "elementary gates" are implemented with carefully tuned RF pulse sequences. Gate costs can be affected not only by direct resource requirements (size, runtime, available frequency channels) but also by considerations of circuit reliability in the context of frequent transient errors (e.g., decoherence of quantum bits). Some gates may be more amenable to error-correction than others, e.g., the CNOT gate and other linear transformations allow for convenient fault-tolerant extensions. In order to abstract away specific technology details, several abstract cost functions have been proposed in the literature. However, their relevance strongly depends on future developments in quantum-circuit technologies.

- *Speed* was defined in [Beckman et al. 1996] to approximate the runtime of a quantum computation on an ion trap-based quantum technology, assuming all laser pulses take equal amounts of time. They observed that the C^{k}NOT gate ($k = 1, 2,$

[10]A Hamiltonian describes time-dependent behavior of a quantum system and can be compared to a set of forces acting on a non-quantum system.

etc.) can be implemented by $2k + 3$ laser pulses. The authors assumed that only one gate can be applied at a time.

- *Number of one-qubit gates and CNOT* (or any other two-qubit gate) is a complexity metric for quantum synthesis algorithms. Since CNOT is a linear gate, the number of one-qubit gates (excluding inverters) needed to express a computation is defined as a measure of non-linearity for a given computation [Shende and Markov 2009].
- *Quantum cost* (QC) is defined as the number of NOT, CNOT, controlled-V and controlled-V† gates required for implementing a given reversible function. These gates can be efficiently implemented in an NMR-based quantum technology by a sequence of electromagnetic pulses [Lee et al. 2006]. Under any other quantum technology, primitive gates can be adapted similarly. For example, while Toffoli needs five gates from the NCV library (two CNOT, two controlled-V, and one controlled-V† gates) it needs exactly six CNOTs and several one-qubit gates under the universal set of one-qubit and CNOT gates (Fig. 6b) [Shende and Markov 2009]. In another example, the Fredkin gate is easier to implement than the Toffoli gate under some quantum technologies [Fei et al. 2002].[11] A single-number cost model, based on the number of two-qubit operations required to simulate a given gate, was used in [Maslov and Saeedi 2011] where costs of both n-qubit Toffoli and n-qubit Fredkin gates (and $n \geq 3$) are estimated as $10n - 25$. QC of a circuit is calculated by a summation over the QCs of its gates.
- *Interaction cost* is the distance between gate qubits for any 2-qubit gate. Quantum circuit technologies with 1D, 2D and 3D interactions exist [Cheung et al. 2007]. Interaction cost for a circuit is calculated by a summation over the interaction costs of its gates.
- *Number of ancillae* and *garbage bits* (ancillae not reset to 0) reflects the limited number of qubits available in contemporary quantum computers.
- *Depth* (or the number of levels) is defined as the largest number of elementary gates on any path from inputs to outputs in a circuit. When any subset of gates can be invoked simultaneously, decreasing circuit depth reduces circuit latency. This assumption is trivial for conventional semiconductor circuits because the gates are manufactured individually and exist at the same time. However, when quantum gates are invoked by electromagnetic pulses, their parallel invocation must clear a number of obstacles — it should be possible to select just the right set of qubits on which the gates are applied, which may require several laser sources and possibly several pre-determined wavelengths. When the parallel gates perform different functions, interference between them may limit achievable parallelism. Practical quantum computers can either apply the same gate to all qubits *or* apply different gates to a small number of qubits.

As pointed out in [Beckman et al. 1996], specific quantum-circuit technologies may entail more involved cost functions where the delay of a gate may depend on neighboring gates. The abstract cost functions introduced above do not capture such effects.

3. GENERATION AND OPTIMIZATION OF REVERSIBLE CIRCUITS

In this section we outline key steps in generation and optimization of reversible circuits, as illustrated in Fig. 7. Algorithmic details will be given in Sections 4 and 5. To implement an irreversible specification using reversible gates, ancillae should be added to the original specification where the number of added lines, their values, and

[11] Fredkin can be constructed using three Toffoli gates by adding one control to each CNOT gate in Fig. 1b. The three Toffolis can then be simplified into two CNOTs around a Toffoli.

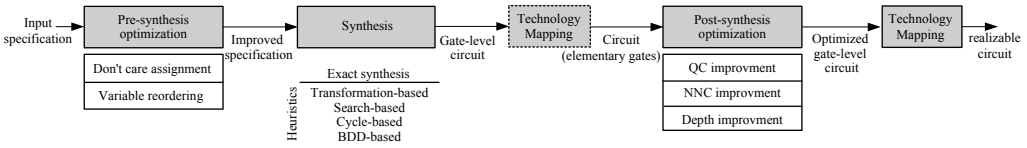


Fig. 7. A general flow used in recent reversible logic synthesis methods.

the ordering of output lines affect the cost of synthesized circuits. This process can be either performed prior to synthesis or in a unified approach during synthesis.

Synthesis seeks reversible circuits that satisfy a reversible specification. It can be performed optimally or heuristically.

- *Optimal* iterated deepening A*-search (IDA*) algorithm was used in [Shende et al. 2003] to find optimal circuits of all 3-input reversible functions. Golubitsky et al. [2010] observed that an optimal realization of some reversible functions can be constructed from an optimal circuit of another function — no need to synthesize all functions independently. For example, optimal circuits for f^{-1} can be constructed by reversing optimal circuits for f. By exploiting such symmetries and using a hashing technique, the authors found optimal circuits for all 4-input permutations. Symbolic reachability analysis [Hung et al. 2006] and Boolean satisfiability (SAT) [Grosse et al. 2009] have been applied to find optimal realizations for reversible functions. These methods mainly formulate the synthesis problem as a sequence of instances of standard decision problems, such as Boolean satisfiability, and use third-party software to solve these problem instances. Only a small number of qubits and gates can be handled by these methods.
- *Asymptotically optimal* synthesis was proposed by Patel et al. [2008] for linear reversible circuits which leads to $\Theta(n^2/\log n)$ CNOT gates in the worst case. Maslov [2007] addressed depth-optimal synthesis of *stabilizer circuits* and proposed a synthesis algorithm that constructs circuits by concatenating $90n + O(1)$ stages, each stage containing only one type of gates (CNOTs or certain one-qubit gates). Asymptotically optimal methods may not produce optimal circuits for specific inputs.

Since most circuits of practical interest are non-linear and too large for optimal synthesis, heuristic algorithms were proposed. The choice of a representation model for reversible functions plays a significant role in developing effective synthesis algorithms. Each model favors certain types of reversible functions by representing them concisely. Synthesis algorithms are developed by detecting such simple cases and decomposing reversible functions into sequences of simpler functions in a given model.

- *Transformation-based* methods [Miller et al. 2003; Maslov et al. 2007] iteratively select a gate so as to make a function's *truth table or RM spectrum* more similar to the identity function. These methods are mainly efficient for permutations where output codewords follow a regular (repeating) pattern.
- *Search-based* methods [Gupta et al. 2006; Donald and Jha 2008] traverse a search tree to find a reasonably good circuit. These methods mainly use the *PPRM expansion* to represent a reversible function. The efficiency of these methods is highly dependent on the number of circuit lines and the number of gates in the final circuit.
- *Cycle-based* methods [Shende et al. 2003; Saeedi et al. 2010a] decompose a given permutation into a set of disjoint (often small) *cycles* and synthesize individual cycles separately. Compared to other algorithms, these methods are mainly efficient

for permutations without regular patterns and reversible functions that leave many input combinations unchanged.

- *BDD-based* methods [Wille and Drechsler 2009; Wille et al. 2010b] use *binary decision diagrams* to improve sharing between controls of reversible gates. These techniques scale better than others. However, they require a large number of ancilla qubits — a valuable resource in fledgling quantum computers.

Several other heuristics do not directly use the discussed representation models. Some reuse algorithms developed for conventional logic synthesis, e.g., the algorithm proposed in [Mishchenko and Perkowski 2002] uses ancillae to convert an optimized irreversible circuit into a reversible circuit. In [Fazel et al. 2007] a circuit is constructed as a cascade of ESOP gates in the presence of some ancillae. Another approach uses abstract group theory to synthesize reversible circuits [Storme et al. 1999; Rentergem et al. 2007; Yang et al. 2006]. However, as of 2011, empirical performance of reported implementations lags behind that of more established approaches. Heuristic synthesis is discussed in Section 4.3, while synthesis of optimal circuits is explored in Sections 4.1 and 4.2.

Post-synthesis optimization. The results obtained by heuristic synthesis methods are often sub-optimal. Further improvements can be achieved by local optimization.

- *Improving gate count and quantum cost*. To improve the quantum cost of a circuit, several techniques attempt to improve individual sub-circuits one at a time. Sub-circuit optimization may be performed based on offline synthesis of a set of functions using pre-computed tables [Prasad et al. 2006; Maslov et al. 2008a], online synthesis of candidates [Maslov et al. 2007; Arabzadeh et al. 2010], or circuit transformations that involve additional ancillae [Miller et al. 2010; Maslov and Saeedi 2011].
- *Reducing circuit depth*. To realize a low-depth implementation of a given function, consecutive elementary gates with disjoint sets of control and target lines should be used to provide the possibility of parallel gate execution. Circuit depth may also be improved by restructuring controls and targets of different gates in a synthesized circuit [Maslov et al. 2008a].
- *Improving locality*. For the implementation of a given computation on a quantum architecture with restricted qubit interactions, one may use SWAP gates to move gate qubits towards each other as much as required. The interaction cost of a given computation can be hand-optimized for particular applications [Fowler et al. 2004a; Kutin et al. 2007; Takahashi et al. 2007]. A generic approach can also be used to either reduce the number of SWAP gates [Saeedi et al. 2011b] or find the minimal number of SWAP gates [Hirata et al. 2011] for a circuit.

Incremental optimization can significantly improve synthesis results, but it cannot guarantee optimality. To illustrate this, consider the NCT-optimal circuit in Fig. 8a [Prasad et al. 2006]. Suppose the pattern is continued by adding one gate at a time until the circuit becomes suboptimal for the function it computes. In the resulting circuit, no suboptimal sub-circuits are formed, and hence no local-optimization method can find a reduction that is available. Section 5 offers additional details on post-synthesis optimization.

Technology mapping. To physically implement a circuit using a given technology, all gates should be mapped (decomposed) into gates directly available in this technology. Such technology mapping can be applied either before post-synthesis optimization or after. Barenco et al. [1995] showed that a multiple-control Toffoli gate in a circuit on n qubits can be mapped into a set of Toffoli gates, with different circuit sizes, depending on how many ancillae are available.

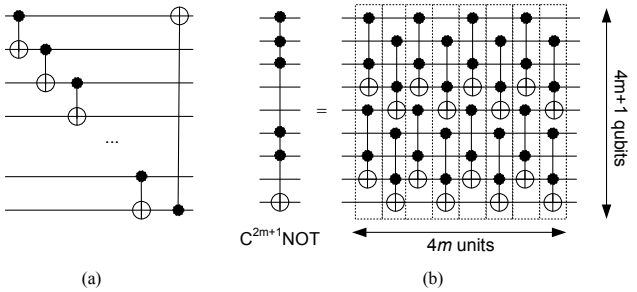


Fig. 8. An optimal circuit used to illustrate limitations of local optimization [Prasad et al. 2006]. If the pattern is continued by adding one gate at a time until the circuit first becomes suboptimal, no local change could make the resulting circuit optimal (a), decomposition of a C^{2m+1}NOT gate [Asano and Ishii 2005], $m = 2^k$, $k \geq 1$. In this figure, $k = 1$ (b).

(1) Without ancilla, $n \geq 3$: A C^{n-1}NOT gate can be simulated by $2^{n-1} - 1$ controlled-V and controlled-V† gates and $2^{n-1} - 2$ CNOTs.
(2) With one ancilla, $n \geq 7$: A C^{n-2}NOT gate can be simulated by $8(n-5)$ Toffoli gates.
(3) With $m - 2$ ancillae, $m \in \{3, 4, \cdots, \lceil n/2 \rceil\}$, $n \geq 5$: A C^mNOT gate can be simulated by $4(m - 2)$ Toffoli gates.

Maslov and Dueck [2003] converted Toffoli gates into (inverse) Peres gates which leads to $32m-96$ and $16m-32$ elementary gates from the NCV library for the cases (2) and (3), respectively. Asano and Ishii [2005] presented a quantum circuit, illustrated in Fig. 8b, to simulate a C^{2m+1}NOT gate on $4m + 1$ qubits, $m = 2^k$, $k \geq 1$, that contains $4m$ units of Toffoli gates. Each unit performs m Toffoli operations simultaneously on $3m$ qubits. By eliminating individual-qubit manipulation, their circuit increases parallelism in quantum circuits at the cost of additional gates.

Maslov et al. [2008a] improved the result of [Maslov and Dueck 2003] by removing redundant controlled-V gates which leads to $12m - 22$ and $24n - 88$ gates for (2) and (3), correspondingly. Fig. 9 illustrates the decomposition of a C^6NOT gate where b-c, d, and e are the results of applying the methods of [Barenco et al. 1995], [Maslov and Dueck 2003], and [Maslov et al. 2008a], respectively. Miller and Sasanian [2010] proposed techniques to reduce the number of elementary gates for C^cNOT, $c \in \{3, ..., 15\}$ assuming $\{1, 2, ..., c - 2\}$ ancillae. Synthesis and post-synthesis optimization methods which consider the underlying gate libraries, e.g., to improve locality or to decrease circuit depth, should also benefit from an internal technology mapping.

4. ALGORITHMS FOR REVERSIBLE CIRCUIT SYNTHESIS

In the following subsections, we discuss exact and asymptotically optimal synthesis methods followed by heuristic algorithms.

4.1. Optimal Methods

For a reversible circuit with n lines, where its optimal realization needs h gates from a library $\mathcal{L}$, an enumerative method may branch h ways on each $\mathcal{L}$-gate. For example, assume that only multiple-control Toffoli gates exist in the library. For this simplified case, an exhaustive method examines $(n \times 2^{n-1})^h$ gates[12] to find an optimal circuit. For $n = 3$, the worst-case circuit needs eight gates from the NCT library. Therefore, only

[12]There are $\binom{n}{1}$ possible NOT gates and $\binom{n}{2}$ possible CNOT gates in which one of its two inputs can be the target output. Hence, the total number of $2\times\binom{n}{2}$ CNOT gates can be obtained. For a (k+1)-bit gate, $k \in (2, 3, \cdots, n-1)$, there are $\binom{n-1}{k}$ possible gates when the target can be the i-th ($i \in [1, n]$) bit. Considering

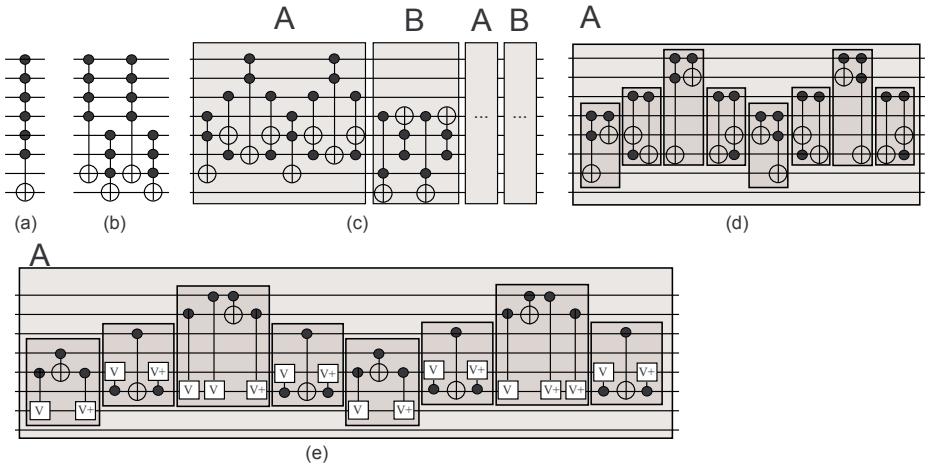


Fig. 9. Decomposition of a C^{6}NOT(a,b,c,d,e,f,h) gate into smaller multiple-control Toffoli gates [Barenco et al. 1995] (b) and Toffoli gates [Barenco et al. 1995] (c), Peres gates [Maslov and Dueck 2003] (d), elementary gates [Maslov et al. 2008a] (e), in the presence of one garbage line.

12^8 different cases should be examined. For $n = 4$, optimal circuits with 15 gates exist [Golubitsky et al. 2010], hence, $2^{75} \simeq 3.8 \times 10^{22}$ different cases should be analyzed by exhaustive search to ensure that a min-cost circuit is found.

3-qubit circuits. Shende et al. performed gate-count optimal synthesis of 3-bit reversible functions by gradually building up a library of optimal circuits for all 8! permutations, rather than by dealing with each permutation individually [Shende et al. 2003]. Noting that every sub-circuit of an optimal circuit is also optimal, they stored optimal circuits with m gates and added one gate at the end of each stored circuit in all possible ways. Those resulting circuits that implement new functions can be added to the library. To lower memory usage when synthesizing a given permutation, instead of examining all optimal circuits with k gates in the library for increasing values of k, the algorithm in [Shende et al. 2003] stops at m ($m \leq k$) gates and seeks circuits with $m + 1$ gates that implement the permutation. In the absence of solutions, it seeks circuits with $m + 2$ gates and so on.

4-qubit circuits. Optimal synthesis of 4-bit reversible functions was investigated in [Prasad et al. 2006], [Yang et al. 2008], and [Golubitsky et al. 2010]. Initially, Prasad et al. [2006] introduced a data structure to represent all 40320 optimal 3-input and about 26,000,000 optimal 4-input reversible circuits with up to six gates from the NCT library. Yang et al. [2008] improved this method where the implementation of a specification on four variables was explored in a search tree based on a bidirectional approach [Miller et al. 2003]. Consequently, over 50% of even 4-bit reversible circuits (approximately one quarter of all possible ones) were optimally realized with up to 12 NOT, CNOT and Peres gates. Golubitsky et al. [2010] offered further improvements. They noted that in an optimal circuit with k gates, the first $\lceil k/2 \rceil$ gates and the last $\lfloor k/2 \rfloor$ gates must also form optimal circuits for respective functions. Hence, they first synthesized all half-sized optimal circuits and stored them in a hash table. The hash table was searched next for finding both halves of any optimal circuit with four in-

all possible bits as the target leads to $n \times \binom{n-1}{k}$ (k+1)-bit gates. Therefore, the total number of gates is $\binom{n}{1} + 2 \times \binom{n}{2} + n \times (\sum_{i \in (2 \cdots n-1)} \binom{n-1}{i}) = n \times 2^{n-1}$.

Table I. The distribution of gate counts in gate-count optimal circuits for all 3- and 4-qubit functions with respect to the NCT library.

Number of gates	3-bit Functions [Shende et al. 2003]	4-bit Functions [Golubitsky and Maslov 2011]
15	0	144
14	0	37,481,795,636
13	0	4,959,760,623,552
12	0	10,690,104,057,901
11	0	4,298,462,792,398
10	0	819,182,578,179
9	0	105,984,823,653
8	577	10,804,681,959
7	10,253	932,651,938
6	17,049	70,763,560
5	8,921	4,807,552
4	2,780	294,507
3	625	16,204
2	102	784
1	12	32
0	1	1
Total	$2^3!$=40,320	$2^4!$ =20,922,789,888,000

puts. Additionally, a simultaneous input/output relabeling (reordering) was applied, and symmetries of reversible functions were used to further reduce the search space. Optimal realization for the inverse f^{-1} of a function f was obtained by reversing an optimal circuit of f. The last two techniques reduce the search space by more than a factor of 48 (i.e., $2 \times 4!$). Running for less than 3 hours on a high-performance server with 16 AMD 2300 MHz processors and 64 GB RAM, Golubitsky et al. [2010] found the distributions of gate-count optimal 4-bit circuits up to 15 gates reproduced in Table I.

Adapting algorithms from formal verification. In order to find optimal circuits for reversible functions with more than four inputs, several sophisticated techniques draw upon algorithms and data structures from the field of formal verification [Hachtel and Somenzi 2000]. Two optimal synthesis approaches for generic reversible and irreversible functions were developed in [Hung et al. 2006] and [Grosse et al. 2009] where the former uses symbolic reachability analysis[13] and the latter applies Boolean satisfiability. In [Hung et al. 2006] a circuit is considered as a cascade of L stages each of which is a 1-qubit or 2-qubit gate from the NCV library. Stage parameters (i.e., gate type and gate qubits) are modeled such that outputs of i-th stage are connected to inputs of $(i+1)$-th stage. In this scenario, a minimal-length circuit is equivalent to the smallest L. In contrast, for a given reversible function f, the algorithm of [Grosse et al. 2009] seeks the availability of a circuit implementing f with a sequence of d multiple-control Toffoli gates. Starting with $d = 1$, d is incremented until a circuit is found. While circuits are modeled in a similar fashion, the method in [Hung et al. 2006] constructs an FSM (Finite State Machine) and employs a SAT solver to find a counter-example. To achieve this, instead of working with L cascaded stages, 2^n parallel FSM instances are generated for truth table rows. The outputs of all 2^n instances at time t are inputs of modules at time $t + 1$. Grosse et al. [2009] used Boolean satisfiability and several common SAT techniques as well as problem-specific information to improve runtime. Optimal circuits with respect to interaction cost can be found similarly [Saeedi et al. 2011b]. To improve runtime when handling large circuits, Wille et al.

[13]Given a finite-state machine described by a sequential circuit and a set of states described by a property, the *reachability problem* asks if the (un)desired states can be reached from the initial state through a sequence of valid transitions.

[2008b] used a generalization of Boolean satisfiability, Quantified Boolean Formula (QBF) satisfiability, and BDDs (Binary Decision Diagrams).

4.2. Asymptotically Optimal Methods

Aaronson and Gottesman [2004] demonstrated that any stabilizer circuit can be restructured into 11 stages of Hadamard (H), Phase (P) and linear reversible circuits (C) in the order H-C-P-C-P-C-H-P-C-P-C. They also proved that the use of Hadamard and Phase provides at most a polynomial-time computational advantage since stabilizer circuits can be simulated by only NOT and CNOT gates. However, even when Hadamard and Phase gates are used, the size of a stabilizer circuit is likely to be dominated by the size of CNOT blocks. We therefore turn our attention to asymptotically optimal[14] synthesis of linear functions.

When reversible functions are captured by (unitary) matrices, each row and each column include a single '1' and '0's elsewhere. A different model proposed in [Patel et al. 2008] is specific to linear circuits and represents CNOT(i, j) by inserting a '1' into the (i, j) element of the identity matrix. This model allows one to cast synthesis of linear circuits as the task of reducing a given matrix (of the function to be synthesized) to the identity matrix by elementary row operations over GF(2). Each row operation corresponds to a CNOT gate, and the sequence of row operations gives a reversible circuit. This task is usually solved using Gaussian elimination, which requires $O(n^2)$ row operations and $O(n^3)$ time. To this end, the input matrix is reduced to an upper triangular matrix by a set of row operations, the resulting matrix is transposed, and this process is repeated on the transposed matrix. To reduce the number of gates, Patel et al. [2008] partition an $n \times n$ matrix into a set of sections each one contains m (e.g., $m = \log_2 n$) columns. To construct an upper-triangular matrix, the algorithm eliminates repeated rows in each section by applying carefully-planned row operations first. Then, diagonal entries are fixed, and Gaussian elimination is used to remove all off-diagonal entries. As in the standard approach, the same scenario is applied to the transposed matrix. This technique reduces the worst-case number of operations (equivalently the size of an n-wire CNOT circuit) to $\Theta(n^2/\log n)$ which is asymptotically optimal. Its runtime is improved to $O(n^3/\log n)$ versus $O(n^3)$ for Gaussian elimination. Maslov [2007] studied the depth (instead of size) of stabilizer circuits where only adjacent qubits can interact. By presenting a constructive algorithm based on Gauss-Jordan elimination, he demonstrated that any stabilizer circuit can be executed in at most $30n + O(1)$ stages composed of only generic two-qubit gates. For the library of CNOT and single-qubit gates, an (asymptotically-optimal) upper bound is $90n + O(1)$.

4.3. Heuristic Methods

Finding an optimal circuit for a given arbitrary-size reversible specification is intractable in general, hence heuristic methods have been developed to find reasonable solutions in practice. In this section, we review those methods that either significantly improved upon prior results or introduced new insights.

Transformation-based methods. Miller et al. [2003] proposed a synthesis method that compares the identity function (I) with a given permutation (F), as illustrated in Fig. 10a, and applies reversible gates to *transform* F into I. To direct the transformation (or select a gate), the complexity metric used is the sum of Hamming distances

[14] An algorithm is asymptotically optimal if it performs at worst a constant factor worse than the best possible algorithm. Formally, for a problem which needs $\Omega(f(n))$ overhead according to a lower-bound theorem, an algorithm which requires $O(f(n))$ overhead is asymptotically optimal. While an such a method cannot find the solution optimally, no algorithm can outperform it by more than a fixed constant factor. On the other hand, other algorithms may find smaller circuits in specific cases, run faster or use less memory.

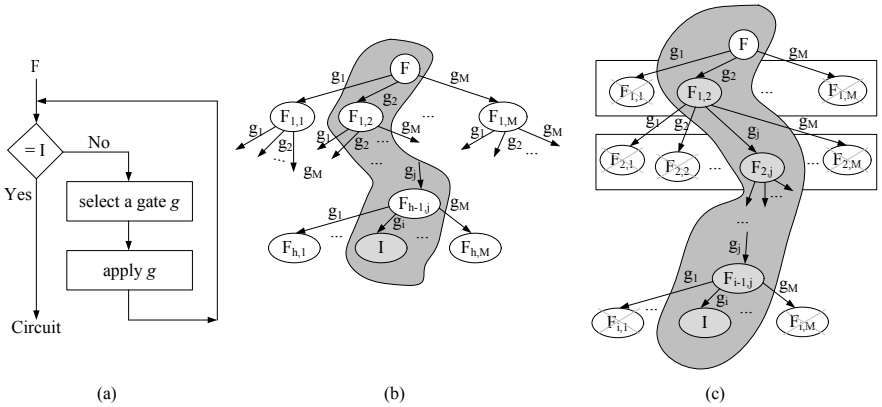


Fig. 10. Outlines of transformation-based algorithms (a), search-based methods (b), and search-based methods directed by a complexity metric (c). In this figure, F is the input permutation, I is the identity function, g_i is a possible gate, M is the maximum number of gates, and $F_{i,j}$ is a permutation which results from applying g_j at the i-th level.

between binary patterns of F and I at each truth table row. The algorithm iterates through the rows of the truth table, looks for differences between F and I, and corrects these differences by applying multiple-control Toffoli gates with positive controls only. This algorithm was improved in [Maslov et al. 2007] where the authors direct synthesis by the complexity of the Reed-Muller spectra instead of the Hamming distance. The algorithm proposed in [Maslov et al. 2007] produces best-known circuits for several families of benchmark functions with regular patterns in their permutations.

Multiple-control Toffoli gates with both positive and negative controls in a column-wise (vs. row-wise as in [Miller et al. 2003]) scenario were used in [Saeedi et al. 2007b]. This algorithm results in circuits composed of complex gates with common targets. Gates that share targets/controls can be further optimized by post-processing [Arabzadeh et al. 2010; Maslov and Saeedi 2011].

Search-based methods. As shown in Fig. 10b the search process can be represented by a tree. One may *search* for an implementation of a function by starting from an initial specification (root of the tree), applying individual gates (to generate branches), and repeating this process on the resulting functions until the identity specification is found in a branch. Given enough memory and time, this method can find a minimal circuit. It is useful when gate-counts and the numbers of inputs/outputs are small. To make this approach practical, one can select only those gates that minimize a specific metric as illustrated in Fig. 10c. For example, in [Gupta et al. 2006] common sub-expressions between the PPRM expansions of multiple outputs are identified and used to simplify the outputs at each stage. Discovered factors are substituted into the PPRM expansions to determine their potential for leading to a solution where the primary objective is gate count (i.e., number of factors) minimization and the secondary objective is gate size (i.e., number of literals in each factor) reduction. To share factors among multiple outputs, candidate factors are selected among common sub-expressions in PPRM expansions. However, there is no guarantee that the resulting PPRM expressions contain fewer terms [Saeedi et al. 2007a]. To relax optimization criteria, instead of evaluating previously substituted factors before new substitutions, Saeedi et al. [2007a] considered all new factors first and proposed a hybrid method that applies the second approach before the first. Donald and Jha [2008] improved the method of [Gupta et al. 2006] to handle gates in the NCTSFP library in a simi-

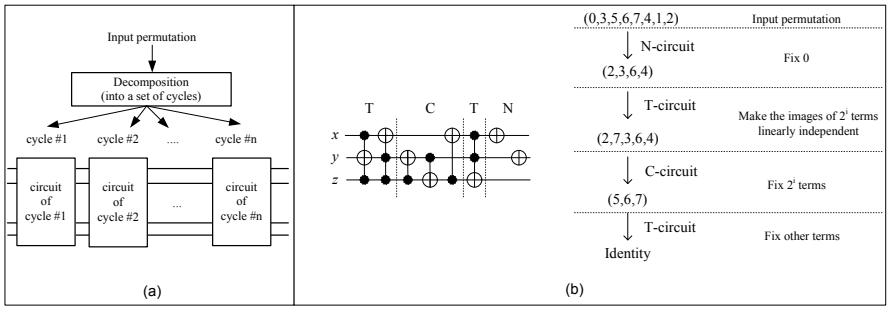


Fig. 11. A general outline of cycle-based methods (a), an example of the T|C|T|N method (b).

lar search-based framework. These algorithms can handle various gate types. Their performance is affected by the number of input bits and the size of resulting circuits.

Cycle-based methods. Instead of working with an entire permutation, one can factor it into a set of *cycles* and synthesize the resulting cycles separately as illustrated in Fig. 11a. This divide-and-conquer approach is particularly successful with reversible transformations that leave many inputs unchanged (sparse transformations).

Shende et al. [2003] proposed an NCT-based synthesis method which applies NOT (N), Toffoli (T), CNOT (C), and Toffoli (T) gates in order (i.e., the T|C|T|N method) to synthesize a permutation. As illustrated in Fig. 11b, in the first C|T|N part, the terms 0 and 2^i of a given function are positioned at their right locations. The last Toffoli circuit fixes the other truth table terms by decomposing the resulting permutation into a set of transpositions. Subsequently, each pair of disjoint transpositions is implemented by a synthesis algorithm separately, and the final circuit is constructed by cascading individual circuits. A similar method was introduced in [Yang et al. 2006] except for working with neighboring 3-cycles, i.e., cycles whose elements differ only in two bits. This technique often produces an unnecessarily large number of cycles. An extension of method from [Shende et al. 2003] described in [Prasad et al. 2006] reduces synthesis cost by applying NOT and CNOT instead of Toffoli in many situations.

Saeedi et al. [2010a] developed k-cycle synthesis, leading to significant reductions in the quantum cost for large cycles, based on seven building blocks — a pair of 2-cycles, a single 3-cycle, a pair of 3-cycles, a single 5-cycle, a pair of 5-cycles, a single 2-cycle (4-cycle) followed by a single 4-cycle (2-cycle), and a pair of 4-cycles — and a set of algorithms to synthesize a given cycle of length less than six [Saeedi et al. 2010a]. Larger cycles are factorized into proposed building blocks. A hybrid synthesis framework was suggested which uses the cycle-based approach for irregular functions in conjunction with the method of [Maslov et al. 2007] for regular functions. The proposed cycle-based method leads to best-known circuits with respect to quantum cost for permutations which have no regular pattern. In addition, the maximum number of elementary gates for any permutation function in [Saeedi et al. 2010b] is less than $8.5n2^n + o(2^n)$, which is the sharpest upper bound for reversible functions so far (the lower bound is $n2^n / \log n$ [Shende et al. 2003]). A more efficient decomposition algorithm was proposed in [Saeedi et al. 2010b] which produces all minimal and inequivalent factorizations each of which contains the maximum number of disjoint cycles. These decompositions are used in a cycle-assignment algorithm based on the graph matching problem to select the best possible cycle pairs during synthesis.

BDD-based methods. Kerntopf [2004] introduced a synthesis algorithm that uses binary decision diagrams (BDDs), and seeks to minimize the number of non-terminal DD nodes. At each step, all possible gates are examined, and the corresponding deci-

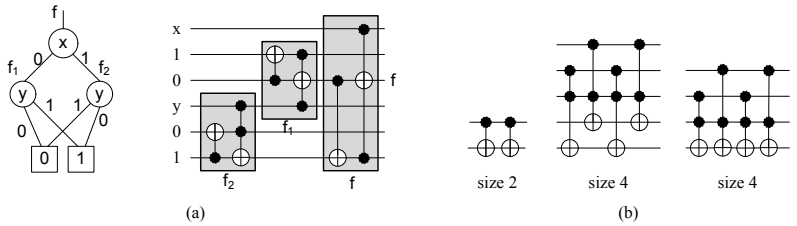


Fig. 12. A BDD-based circuit synthesis technique by example [Wille and Drechsler 2009] (a), templates of sizes 2 and 4 [Maslov et al. 2007] (b).

sion diagrams are constructed. The gates that minimize the complexity metric are selected and further analyzed by repeating the same process. Wille and Drechsler [2009] introduced a different algorithm that starts by constructing a BDD. Each BDD node is substituted by a cascade of reversible gates as shown in Fig. 12a. Node sharing due to reduction rules in ROBDDs can cause gate fanout which is prohibited in reversible logic. To overcome this obstacle, the algorithm adds constant bits to emulate fanout — in the worst case, for each BBD node, a new constant line may be added. While this algorithm leads to a good reduction in both quantum cost and runtime, many constant and garbage bits are added which makes the results impractical for quantum computers with a limited number of qubits. Wille et al. [2010b] reduced the number of lines by a post-processing technique where garbage lines are merged with appropriate constant lines. Although BDD-based techniques for reversible synthesis scale better than most other approaches, the large number of ancillae they generate makes the results difficult to use in practice, and the effort to consolidate ancillae can be substantial.

5. POST-SYNTHESIS OPTIMIZATION

To improve the results of synthesis algorithms, several optimization methods consider connected subsets of gates (sub-circuits) in a given circuit. Such sub-circuits are analyzed one by one and replaced by equivalent (smaller) sub-circuits to improve cost. This sub-circuit replacement approach can leverage earlier-discussed techniques to improve large circuits using *peephole optimization* with linear runtime [Prasad et al. 2006].

5.1. Quantum Cost Improvement

Equivalent sub-circuits can be found using either windowing or sub-circuit optimization and replacement [Prasad et al. 2006; Maslov et al. 2007; Maslov and Saeedi 2011].

Library-based optimization. Prasad et al. [2006] proposed an algorithm that uses a large database of optimal circuits and seeks sub-circuits that can be replaced by smaller equivalent sub-circuits. In practice, the stored sub-circuits are likely to be very limited in size. Prasad et al. [2006] introduced a compact data structure that can store all 3-bit reversible circuits and many 4-bit circuits with less than six gates. A windowing strategy proposed in [Prasad et al. 2006] to identify contiguous sub-circuits can reorder some gates (without changing the overall functionality) to assemble larger 4-bit sub-circuits. The functionality of the sub-circuit found is computed, and a database look-up is performed to find an optimal circuit that implements the same functionality. The sub-circuit is replaced if this improves cost. Originally, this algorithm was applied to optimize reversible circuits composed of NOT, CNOT and Toffoli gates, but it can work with other gates as well. Such optimizations rely heavily on a database of optimal implementations and an efficient windowing strategy.

Each circuit stored in a library can be viewed as a rule that simplifies any other circuit that computes the same function. For example, pairs of inverters, pairs of CNOTs

and pairs of Toffoli gates cancel out because the same function can be computed by an empty circuit. To reduce the size of a library, such rules can be generalized by local circuit transformations, leading to more compact rule sets.

Transformation rules and template-based optimization. The work performed in [Iwama et al. 2002] introduced the idea of local transformation of reversible circuits. While the main purpose of this work was not post-synthesis optimization, its results were extended by other researchers to improve circuit cost. The authors defined a canonical form for circuits in the NCT library, and introduced a complete set of rules to transform any NCT-constructible circuit into its canonical form, which may or may not be compact.

The concept of applying a rule set was extended in [Miller et al. 2003] where the authors introduced several transformation rules based on a set of predefined patterns called *templates*. A template T is a reversible circuit that implements the identity function, which contains m gates g_1, g_2, $\cdots$, g_m. For a circuit with multiple-control Toffoli and Fredkin gates, consider the first k ($k > m/2$) gates of T (i.e., g_1, g_2, $\cdots$ g_k). Suppose that these gates are found in a reversible circuit in sequence. It can be verified that the set of $m - k$ gates g_m, $\cdots$, g_{k+2}, g_{k+1} can be applied instead of the initial g_1, g_2, $\cdots$ g_k gates to reduce the gate count from k to $m - k$.[15] The authors showed that applying the template matching method (called *template application algorithm*) with two- and three-input templates only can improve the circuits.

In [Maslov et al. 2005b], template matching with up to six gates was used in post-synthesis optimization. The authors showed that there are 0, 1, 0, 1, 1, and 4 templates for gates with 1, 2, 3, 4, 5, and 6 gates, respectively. Their analysis shows that these seven templates comprise a complete set of templates of size < 6, for < 4 inputs. Similarly, the Toffoli-Fredkin templates were explored in [Maslov et al. 2005a] where the authors showed that there are 0, 1, 0, 3, 1, and 1 Toffoli-Fredkin templates for gates with 1, 2, 3, 4, 5, and 6 gates. Toffoli templates were extended in [Maslov et al. 2007] by the addition of all templates of size 7 (five templates) and a set of templates of size 9 (four templates). Fig. 12b shows templates of sizes 2 and 4. In addition, the template application algorithm was enhanced leading to two templates of size 4 (vs. 1) and three templates of size 6 (vs. 4). Saeedi et al. [2011b] extended the templates to work with up to three SWAP gates. Template-based optimizations can be time-consuming, but scale to large circuits due to their local nature [Maslov et al. 2007]. One can restrict template application to small subsets of gates and lines to improve runtime. Such post-processing can be used in peephole optimization with guaranteed linear runtime [Prasad et al. 2006].

Arabzadeh et al. [2010] proposed a set of simplification rules in terms of positively and negatively controlled Toffoli gates. To optimize a sub-circuit which has gates with identical target as illustrated in Fig. 13a, a C^{n-1}NOT gate is represented by a Boolean expression with $n - 1$ inputs and one output where gate controls act as the inputs and the target behaves as the output. Next, this gate fills one cell of a Karnaugh map (K-map) of size n (i.e., $n - 1$ inputs, one output). To extract a simplified circuit, one can use a K-map cell clustering similar to the one used in irreversible logic. The authors showed that each cell with the value 1 can be used in an odd number of groups and each cell with the value 0 can be used in an even number of groups. Some templates in [Maslov et al. 2005b], e.g., the ones in Fig. 12, can be regenerated by applying a set of simplification rules from [Arabzadeh et al. 2010]. This simplification approach is suitable for methods that generate subsequent gates on the same target line [Mishchenko and Perkowski 2002; Saeedi et al. 2007b]. An optimization in [Soeken et al. 2010b]

[15] If $g_1 \cdots g_k g_{k+1} \cdots g_m = I$, then $g_1 \cdots g_k = g_m^{-1} \cdots g_{k+1}^{-1}$. For self-inverse gates (Toffoli, Fredkin), $g^{-1} = g$.

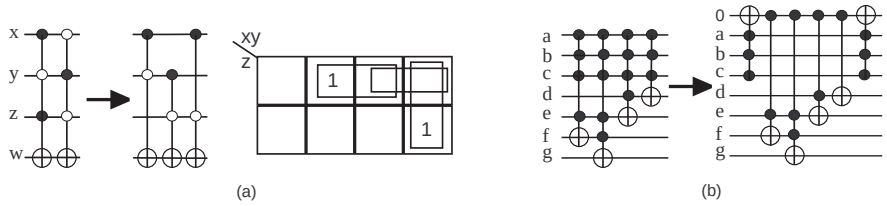


Fig. 13. K-map-based optimization [Arabzadeh et al. 2010] (a), ancilla insertion [Miller et al. 2010] (b).

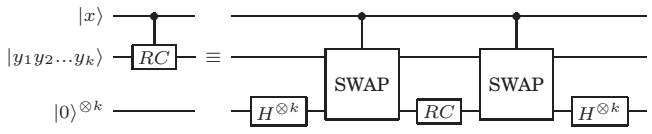


Fig. 14. Circuit equivalence used in [Maslov and Saeedi 2011].

uses a window to select potential sub-circuits first. Then, re-synthesis, exact synthesis and template matching methods are applied to improve the selected sub-circuits.

Qubit insertion. Circuits can be simplified by adding ancillae. A well-known example is the implementation of the n-bit multiple-control Toffoli gate discussed in Section 3. Another example, the generic algorithm in [Miller et al. 2010] searches sub-circuits with a set of shared controls C. Their gates are simplified by removing controls in C. Two identical multiple-control Toffoli gates are inserted before and after the simplified sub-circuit as illustrated in Fig. 13b where their controls are the qubits in C and targets are on the zero-initialized line. This modification produces an equivalent but smaller circuit if the cost of added gates is smaller than that of removed controls in the multiple-control gates. This idea was further extended in [Miller et al. 2010] to add multiple ancillae.

To compute a Boolean function by a quantum circuit, it is common to use only reversible (non-quantum) gates. However, the use of quantum gates offers more freedom and may facilitate smaller circuits in some cases. Maslov and Saeedi [2011] proposed a circuit optimization that uses quantum Hadamard gates and therefore ventures beyond the Boolean domain. For a reversible circuit RC and $|00...0\rangle$ ancillae, they consider the transformation $|x\rangle\,|00...0\rangle \mapsto RC\,|x\rangle\,|00...0\rangle$ with at most n ancilla for n primary inputs in the original reversible circuit. The ancillae are prepared by a layer of Hadamard gates, as shown in Fig. 14. Sets of adjacent gates with shared controls are identified. Since $H^{\otimes k}\,|00...0\rangle$ is a 1-eigenvector of any 0-1 unitary matrix RC, applying RC to this eigenvector does not modify the state. After that, the shared controls are removed from the gates involved. The values are transferred to the ancillae by applying a set of Fredkin gates, and returned to the main qubits by reapplying the same set of Fredkin gates in the reverse order. This optimization is applied opportunistically wherever it improves circuit cost. It is particularly suitable for reversible circuits with many complex gates which can be easily reordered, such as those produced in [Mishchenko and Perkowski 2002].

5.2. Reducing Circuit Depth

Parallel circuits speed up computation and can tolerate smaller coherence times.[16] Maslov et al. [2008a] introduced a level compaction algorithm to reduce circuit level (or depth) of synthesized circuits by employing templates. To this end, a greedy algorithm was proposed which assigns an undefined level to all gates initially. Next, for each level i the leftmost gate with an undefined level is examined to verify whether this gate can be executed at level i or not. This process is continued until the algorithm finds no gate for execution at the i-th level. Next, a set of templates is applied, to change the control and target lines of different gates, and the level assignment process is repeated with the hope of improving circuit depth. Finally, i is incremented and other gates are examined similarly. While the proposed method is useful for level compaction, its efficiency can be improved by applying a more efficient gate selection method.

5.3. Improving Locality

Quantum-circuit technologies often require that each gate involve only geometrically adjacent qubits (in a particular physical layout). Given a fixed number of qubits, a quantum architecture can be described by a simple connected graph $G = (V, E)$, where the vertices V represent qubits and edges E represent adjacent qubit pairs where gates can be applied [Cheung et al. 2007]. A complete graph, K_n, expresses the absence of constraints. The LNN (Linear Nearest Neighbor) architecture corresponds to a graph with n vertices $v_1, \cdots, v_n$ where an edge exists between only neighboring vertices v_i and v_{i+1} for $1 \leq i < n$. Several systems of trapped ions [Häffner et al. 2005], liquid NMR (e.g., [Laforest et al. 2007]), and the original Kane model [Kane 1998] have been designed based on the interactions between linear nearest neighbor qubits. Two-dimensional square lattices (2DSL) corresponds to a graph on a two-dimensional Manhattan grid where only four neighboring qubits can interact. The relevant proposals for 2DSL include arrays of trapped ions [Häffner et al. 2005], Kane's architecture [Skinner et al. 2003], and Josephson junctions [Douçot et al. 2004]. The three-dimensional square lattices (3DSL) model is a set of stacked 2D lattices where a qubit can interact with six neighboring qubits. 3DSL is less restrictive, but suffers from the difficulty of controlling 3D qubits. The architecture proposed in [Pérez-Delgado et al. 2006] relies on the 3DSL model. Cheung et al. [2007] introduced other architectures including the Star architecture with one vertex of degree $n-1$ connected to all other vertices, and the Cycle (C_n), which is LNN with one extra interaction between the first and last qubit. The k-th power of the graph G, denoted by G^k such as LNN^k, is the graph over the same vertex set (of G) with edges representing paths of length k in G.

SWAP insertion. A naive method to satisfy (architectural) qubit-interaction constraints is to use SWAP gates in front of an improper gate g to 'move' the control (target) line of g towards the target (control) line as much as required. Subsequently, SWAP gates should be added to restore the original ordering of circuit lines. This process can be repeated for all gates. More efficient circuits were found in application-specific studies that explored the physical implementations of the quantum Fourier transform [Takahashi et al. 2007; Maslov 2007], Shor's factorization algorithm [Fowler et al. 2004a; Kutin 2007], quantum addition [Choi and Van Meter 2008], and quantum error correction [Fowler et al. 2004b] for the LNN architectures. Researchers considered the impact of LNN constraints on the synthesis of general quantum/reversible circuits in [Shende et al. 2006] where their number of gates was increased by almost an

[16]The study of parallel quantum algorithms has attracted attention in complexity theory too. NC_i is the class of decision problems solvable by a uniform family of Boolean circuits, with polynomial size, depth $O(\log^i(n))$, and fan-in 2. QNC^0 is the class of constant-depth quantum circuits without fanout gates. The question whether $\mathsf{P} \subset \mathsf{NC}_i$ or $\mathsf{P} \subset \mathsf{QNC}^0$ is open.

order of magnitude, and in [Möttönen and Vartiainen 2006] and [Saeedi et al. 2011a] where their numbers of CNOT gates were increased by at most a factor of < 2. Cheung et al. [2007] discussed the translation overhead for converting an arbitrary circuit from one architecture to another one. Particularly, they showed that translating a circuit from K_n to Star, LNN, 2DSL, and 3DSL requires $O(n)$, $O(n)$, $O(\sqrt{n})$, and $O(\sqrt[3]{n})$ overhead, respectively. Converting Star, 2DSL, and 3DSL, LNN^k, C_n, and C_n^k to LNN requires $O(1)$, $O(\sqrt{n})$, $O(\sqrt[3]{n^2})$, $O(k)$, $O(1)$, and $O(1)$ overhead, respectively. Most importantly, Star is the weakest architecture among those considered, e.g., the overhead of converting a circuit from LNN to Star is $O(n)$.

SWAP optimization. To adapt circuits to restricted architectures, synthesis algorithms can minimize the number of elementary gates or the SWAP gates. In this context, exact and heuristic synthesis algorithms as well as post-synthesis optimization methods can be applied. Unlike optimal methods, heuristic post-synthesis optimizations scale well to large functions. Template matching for SWAP reduction and reordering strategies, *global* and *local* reordering, were introduced as powerful tools for SWAP reduction in [Saeedi et al. 2011b]. In global reordering, lines with the highest interaction impact are sequentially chosen for reordering and placed at the middle line. This procedure is repeated until the cost cannot be reduced. In contrast, local reordering traverses a given circuit from inputs to outputs and adds SWAP gates only in front of each non-local gate, but not after. Instead, the resulting ordering is used for the rest of the circuit. This process is repeated until all gates are traversed, as illustrated in Fig. 15. Similar reordering scenarios were applied by hand to reduce the number of SWAPs in specific circuits, e.g., in [Takahashi et al. 2007] for QFT.

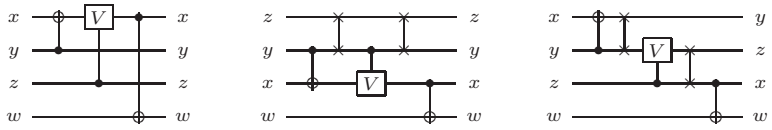
Ensuring the minimal possible number of SWAP gates. For a qubit set $\{x_1, x_2, \cdots, x_n\}$, assume that 1st, 2nd, ..., and n-th qubits should be placed at $C(x_1)$, $C(x_2)$,..., and $C(x_n)$ positions (C is the transformation function) to make a gate local. Hirata et al. [2011] showed the number of SWAP gates necessary for this purpose is at least the number of pairs in the set $S = \{(x_i, x_j) | C(x_j) < C(x_i), i < j\}$ and a bubble sort generates this minimum number of SWAPs for each gate. To find the minimal number of SWAP gates for a given circuit, all possible qubit orderings can be exhaustively searched. However, the efficiency of this approach is limited by the large search space. On the other hand, for two qubits positioned at the locations i_1 and i_2 ($i_2 \geq i_1$), only those qubits that are placed between them need to be considered (i.e., $(i_2 - i_1)!$ permutations instead of all $n!$ permutations) [Hirata et al. 2011]. To further improve runtime, Hirata et al. [2011] considered only $(i_2 - i_1)$ permutations for each gate and analyzed only w consecutive gates instead of considering all possible gates as performed by an exhaustive method. Applying the techniques of [Hirata et al. 2011] with $w = 10$ on the 8-qubit AQFT$_5$ circuit[17] improved hand-optimized results of [Takahashi et al. 2007]. The cost of the AQFT circuit was further optimized by templates introduced in [Saeedi et al. 2011b]. Minimizing AQFT circuits is an open challenge.

Key synthesis and optimization algorithms are compared in Table II.

6. BENCHMARKS AND SOFTWARE TOOLS

To analyze the effectiveness of reversible-logic synthesis algorithms, a variety of benchmark functions are available. Maslov [2011] developed and has been maintaining the

[17]The Quantum Fourier Transform plays a key role in many quantum algorithms. As the number of input qubits grows, QFT needs exponentially smaller phase shifts, which complicates its physical implementation. Therefore, the Approximate Quantum Fourier Transform (AQFT$_m$) was defined by circuits created from QFT except that all phase shift gates R_p with phase $2\pi/2^p$ are ignored for $m > p$ (m is the approximation parameter). While a QFT of size n requires O(n^2) gates to implement, Cheung [2004] showed that AQFT$_m$, $m = \log_2 n$, with O($n \log_2 n$) gates achieves almost the same accuracy level.



(a) Original circuit (b) Global reordering (c) Local reordering

Fig. 15. Global and local reordering [Saeedi et al. 2011b].

Table II. Synthesis and optimization algorithms for reversible circuits.

SYNTHESIS METHOD	FEATURES	LIMITATIONS	LIBRARY	METRIC
[Shende et al. 2003]	Heuristic synthesis Fast No garbage	Circuit dependency	NCT	QC
[Prasad et al. 2006]	Optimization Fast No garbage	Library dependency Local optimum only Circuit dependency Windowing strategy	NCT	QC
[Gupta et al. 2006]	Heuristic synthesis No garbage	Limited scalability	NCT	QC
[Hung et al. 2006]	Optimal synthesis	Limited scalability	NCV	QC
[Maslov et al. 2007]	Heuristic synthesis Optimization	Large runtime Function dependency Local optimum only Windowing strategy	NCT	QC
[Maslov et al. 2008a]	Optimization Fast	Windowing strategy Local optimum only	NCT	Depth
[Donald and Jha 2008]	Heuristic synthesis No garbage	Limited scalability	NCTSFP	QC
[Grosse et al. 2009]	Optimal synthesis	Limited scalability	NCT	QC
[Wille and Drechsler 2009]	Heuristic synthesis Fast, scalable Compact circuits	Numerous ancillae	NCT	QC
[Wille et al. 2010b]	Optimization Fast	Local optimum only Circuit dependency Windowing strategy	NCT	Ancilla
[Arabzadeh et al. 2010]	Optimization Fast	Local optimum only Circuit dependency Windowing strategy	NCT	QC
[Saeedi et al. 2010a]	Heuristic synthesis Fast No garbage	Function dependency	NCT	QC
[Saeedi et al. 2011b]	Optimization Fast	Local optimum only Circuit dependency	Any	Locality
[Hirata et al. 2011]	Optimization	Local optimum only Circuit dependency	Any	Locality
[Maslov and Saeedi 2011]	Optimization Fast	Local optimum only Circuit dependency	Any	QC

Reversible Logic Synthesis Benchmarks Page which offers the widely-used benchmark functions for reversible logic and their best-known circuits (as communicated to the maintainer). `RevLib` introduced in [Wille et al. 2008a] is not limited to best-known circuits and, in addition to some results from [Maslov 2011], includes a variety of sub-optimal circuits. The open-source toolkit `RevKit` [Soeken et al. 2010a] includes several utilities and implements algorithms for reversible circuit synthesis. In [Wille et al. 2010a], a programming language was proposed to specify a reversible transformation from which a compiler can generate reversible circuits. A circuit browser, `RCViewer`, was developed by Scott and Maslov in 2003, and later described in [Maslov 2011]. An improved version, `RCViewer+`, was reported in [Arabzadeh and Saeedi 2011]. RevKit and RCViewer+ support a number of features — circuit visualization and cost anal-

ysis, equivalence checking, and circuit diagram plotting using LaTeX q-circuit format (http://www.cquic.org/Qcircuit/).

The most common reversible benchmark families are as follows.

- **Reversible functions with known optimal circuits** include all 3-input [Shende et al. 2003] and 4-input reversible functions [Golubitsky et al. 2010].
  - *4-bit functions with maximal gate count*: This set introduced by Golubitsky and Maslov [Maslov 2011] contains all 4-bit functions whose optimal implementations use 15 gates (largest number possible).
  - *Gray code transforms*: The N-bit transform `GraycodeN` converts binary-coded integers to Gray-coded integers. As this function is GF(2)-linear, an optimal circuit requires only CNOT gates: CNOT(b,a) CNOT(c,b) .... CNOT(z,y) for qubits a, b, ..., z. Several heuristics [Gupta et al. 2006; Maslov et al. 2007; Saeedi et al. 2010a] produce optimal circuits for this family of functions.

- **Arithmetic functions** have applications in quantum algorithms [Childs and van Dam 2010]. In conventional circuit design, 32-bit and 64-bit arithmetic circuits are of significant interest because they are used in word-level CPUs. Sophisticated optimizations have been developed for such special cases [Dimitrov et al. 2011]. However, no such *standard sizes* have been established for reversible circuits and applications in quantum computing suggest that such standardization is highly unlikely. Therefore, the design of arithmetic circuits focuses on scalable benchmarks and synthesis algorithms rather than a handful of super-optimized circuits. Another distinction from conventional logic circuits is that (as of 2011) we are unable to motivate studies in reversible implementations of floating-point arithmetic.
  - *Adders*: The function `nbitadder` introduced in [Feynman 1986] has two n-bit inputs A and B and one $(n+1)$-bit output $A+B$. Quantum circuits for elementary arithmetic operations are important for the implementation of Shor's factorization algorithm. With one ancilla, a quantum circuit with depth $2n+4$ and size $9n-8$ was proposed in [Cuccaro et al. 2005]. A variant method yields a circuit with size $6n+1$ and depth $6n+1$. Takahashi et al. [2010] proposed a quantum circuit with depth $5n-3$ and size $7n-6$ for `nbitadder` with no ancilla and an O(d(n))-depth O(n)-size quantum circuit with O($\frac{n}{d(n)}$) ancillae where $d(n) = \Omega(\log n)$.
  - *Modulo adders*: The function `modNadder` has $2\lceil \log_2 N \rceil$ inputs/outputs where for each codeword, the input is a pair of modulo-N numbers A and B, while the output is the pair of modulo-N numbers (A, $A+B \mod N$). As of 2011, the best results for `modNadder` functions were obtained by [Maslov et al. 2007]
  - *Galois field multipliers*: The Galois field multiplication function `gfp^mmult` [Cheung et al. 2009] has $2m\lceil \log_2 p \rceil$ inputs and $m\lceil \log_2 p \rceil$ outputs. It computes the field product of two GF(p^m) elements, a and b. GF(p^m) is used in a quantum polynomial-time algorithm that computes discrete logarithm over an elliptic curve group, and it has applications in quantum cryptography. An O(m)-depth multiplication circuit for GF(2^m) targeted for an LNN architecture was proposed in [Cheung et al. 2009].
  - *Divisibility checkers*: The function `NmodK` has N inputs and a single output. Its output is 1 for those codewords that are divisible by the integer K. As of 2011, the best results for `NmodK` functions were obtained by [Maslov et al. 2005b].

- **Hard benchmarks** are mainly proposed to stress-test the existing synthesis algorithms. These functions may be produced from hard benchmarks developed for conventional logic synthesis.
  - *Hidden weighted-bit function*: The function `hwbN` has N inputs/outputs where the input codeword is cyclically shifted by the number of ones it has. The conventional HWB function returns the value of the input bit, indexed by the number of ones (mod n), and all of its ROBDDs have exponential size [Bollig et al. 1999]. Markov and Maslov showed that `hwbN` functions can be implemented with a polynomial cost $O(n \log^2 n)$ if a logarithmic number of garbage bits $\lceil \log n \rceil + 1$ is available [Maslov 2011]. Efficient synthesis with no garbage bits remains open. Known circuits for `hwb` functions with no ancilla exhibit exponential number of gates. As of 2011, the best results for medium-size `hwbN` functions with no ancilla were obtained by applying the method of [Saeedi et al. 2010a].
  - *Reversible variants of high-complexity functions*.
    - *Computation of N-th prime*: The function `nth_primeK_inc` introduced as a reversible benchmark in [Maslov 2011] has K inputs/outputs. For an input value n, this function returns the n-th prime, as long as this prime may be written using at most K bits. The algorithm in [Lagarias and Odlyzko 1987] runs in exponential time, and no poly-time circuits or algorithms are known as of 2011. The smallest circuits to date are shown in [Saeedi et al. 2010a]. For a simpler problem — primality testing — polynomial circuits are proven to exist, but no practical constructions are known as of 2011 [Markov and Roy 2003].
    - *Computation of the matrix permanent*: The function *permanent* NxN introduced by Maslov as a reversible benchmark has N^2 inputs and $\lceil \log(N!) \rceil$ outputs. It computes the permanent of a 0-1 matrix.[18] There is strong evidence that no polynomial-time non-quantum algorithm exists for this computation [Jerrum et al. 2004].

Other reversible functions considered as benchmarks [Maslov 2011; Wille et al. 2008a; Gupta et al. 2006; Grassl 2003] include Hamming coding functions and quantum error-correcting codes.

7. CONCLUSION AND FUTURE DIRECTIONS

Reversible logic circuits have been studied for at least 30 years [Toffoli 1980], with several different motivations in mind — from low-power computing and bit-twiddles in computer graphics algorithms, to photonic circuits and quantum information processing. Synthesis of reversible logic circuits is typically partitioned into (i) a pre-synthesis optimization that revises the specification, (ii) synthesis per se, (iii) post-synthesis local optimization, and (iv) technology mapping that reflects specific limitations of a given implementation technology.

Despite significant progress in reversible logic synthesis, a number of open challenges remain — some are in the domain of reversible circuits and others in the broader domain of quantum information processing. In particular, existing reversible synthesis techniques do not perform well on important benchmarks such as arithmetic functions — they produce circuits that are much larger compared to known solutions.

- **Traditional reversible logic synthesis.**

[18] The permanent of an $n \times n$ matrix $A = (a_{i,j})$ is defined as $\mathrm{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}$ where S_n is the symmetric group and σ is a permutation in S_n. For example, for $N = 2$ the permanent is $a_{1,1}a_{2,2} + a_{1,2}a_{2,1}$.

- Scalable synthesis of general reversible functions targeting different cost models and gate libraries without significant overhead.
- Technology mapping (see Section 3) for specific applications, gate libraries, and cost models.
- Optimal (or efficient) synthesis of reversible functions useful in specific applications, e.g., quantum algorithms such as Shor's number-factoring.
- Efficient synthesis algorithms for T-constructible permutations.

• **Lower and upper bounds, and worst-case reversible functions.**

- Sharper lower and upper bounds on the number of elementary gates for reversible functions. Current lower bound is $n2^n/\log n$ [Shende et al. 2003] and upper bound is $8.5n2^n + o(2^n)$ [Saeedi et al. 2010a].
- Lower and upper bounds on the number of elementary gates for T-constructible reversible functions.
- Super-linear lower bounds on the size and depth of NC-circuits (and stabilizer circuits) for specific functions [Aaronson and Gottesman 2004].
- The minimal number of CNOT gates required for the implementation of an n-qubit Toffoli (or other useful gates such as Fredkin) gate with and without ancillae. Without ancilla, the number of CNOT gates is $\Theta(n^2)$, and $\Theta(n)$ gates are sufficient when at least one ancilla is available [Barenco et al. 1995].

• **Optimization of quantum circuits.**

- Synthesis of circuits with provably minimum size (and depth) for stabilizer (or GF(2)-linear) operators [Aaronson and Gottesman 2004].
- Small quantum circuits for permutation functions [Maslov and Saeedi 2011].

In addition to these rather specific challenges, entirely new concepts and techniques may be discovered for representing reversible functions and synthesizing reversible circuits. Further considerations for future research are summarized below.

Keeping applications in mind. Given that reversible logic circuits today are largely motivated by quantum, nano and photonic computing, we note that these novel computing paradigms promise improvements in rather narrow circumstances, while suffering from serious general drawbacks. For example, quantum algorithms are likely to be handicapped by size limitations, as well as quantum noise and decoherence, but offer polynomial-time algorithms for certain problems where conventional computers currently spend exponential time [Nielsen and Chuang 2000]. Therefore, it does not necessarily make sense to study reversible versions of every conventional circuit. Aside from trying to stress-test logic synthesis tools, specific reversible circuits must be motivated by applications. For example, reversible adders and modular multipliers have been motivated by Shor's quantum number-factoring algorithm [Beckman et al. 1996; Van Meter and Itoh 2005; Markov and Saeedi 2012; 2013] which leverages unique properties of quantum circuits.

Sequential reversible computation has been studied as early as in the 1980s in [Toffoli 1980], but research on this topic is still lacking sufficient motivation. In conventional circuits, sequential elements are clocked, but reversible clocking has not been considered (and may not make sense, since this is not a logic operation). This undermines considerations of low power for sequential reversible computation, as clocking and clocked elements consume a large fraction of energy used by CMOS circuits. Most uses of reversible transformations in cryptography, DSP and computer graphics are combinational in nature. Most quantum computers use stationary qubits and apply gates to these qubits, unlike CMOS circuits where gates are stationary and signals traverse the circuits. In the context of stationary qubits, quantum (and reversible) circuits already have significant sequential semantics [Morita 2008], and there is no need

for dedicated sequential elements. Algorithms for the design, analysis and verification of Quantum Finite Automata [Moore and Crutchfield 2000] may be of some interest, if sufficiently motivated by applications.

Verification of reversible circuits is important because circuit optimization algorithms and software tools have become so complex that subtle bugs are very likely. Fortunately, equivalence-checking techniques for conventional combinational circuits can be applied by converting CNOT gates to XORs, Toffoli gates into ANDs and XORs, and so on (Fig. 1a). Additional efficiency improvements can be obtained by exploiting reversibility as illustrated in [Wille et al. 2009; Yamashita and Markov 2010]. Verification of non-Boolean quantum circuits is more challenging and, in general, appears as hard as quantum simulation [Viamontes et al. 2007; Yamashita and Markov 2010].

Circuit test is vital to check if a circuit works as expected. Efficient test is critical for mass-production facilities, whereas laboratory experiments emphasize precision. Test techniques are sensitive to dominant fault types, but fortunately CMOS circuits can be tested reasonably well assuming only stuck-at fault models [Bushnell and Agrawal 2000]. Given that CMOS is not the dominant technology for reversible circuits, the use of stuck-at fault models in this context may be unjustified, or at least requires explicit justification. An example reversible fault model is given in [Polian et al. 2005], where circuit test is performed for missing gates.[19] In the case of quantum circuits, test is particularly complicated because measurements produce nondeterministic results. Therefore, quantum-computing experiments are typically verified using tomography [Altepeter et al. 2005], i.e., plotting an entire distribution of possible outcomes.

Error-detection and fault-tolerance techniques are motivated for circuit technologies that are likely to experience transient faults, which is the case with quantum circuits. Like circuit test, these techniques are heavily dependent on fault models and measurement, and naive attempts to model quantum faults and error-detection by Boolean techniques lead to nonsensical results. Fault-tolerant quantum computing is an extensively developed branch of quantum information processing, and its basics are introduced in standard textbooks [Nielsen and Chuang 2000].

Quantum-logic synthesis deals with general unitary matrices and is more challenging than reversible-logic synthesis. As of 2011, the most compact circuit constructions use $\frac{23}{48}4^n - \frac{3}{2}2^n + \frac{4}{3}$ CNOTs [Shende et al. 2006; Möttönen and Vartiainen 2006] and $\frac{1}{2}4^n + \frac{1}{2}2^n - n - 1$ one-qubit gates [Bergholm et al. 2005]. The sharpest lower bound on the number of CNOT gates is $\lceil \frac{1}{4}(4^n - 3n - 1) \rceil$ [Shende et al. 2004]. Different trade-offs between the number of one-qubit gates and CNOTs are explored in [Saeedi et al. 2011a]. Future research directions include simultaneous reduction of CNOT and one-qubit gates [Shende and Markov 2009], sharper lower bounds on the number of one-qubit and CNOT gates, and consideration of circuit depth, perhaps with ancillae.

Physical layout and optimization of quantum circuits are crucial to map circuit qubits into physical qubits. Currently, the layout of quantum circuits is hand-optimized when preparing laboratory experiments, but automated techniques are required to systematically accomplish this task as the capacity of quantum computers increases. In [Maslov et al. 2008b], the authors proposed a heuristic for the placement problem by optimizing qubit-to-qubit interaction and showed that the problem of mapping circuit qubits to physical qubits is NP-complete.

[19] In ion-trap and NMR quantum computers, gates are effected by RF pulses. If the wavelength drifts too far from the desired value, the gate will not be applied. This situation illustrates *stuck-at-0 faults* on controls of CNOT and Toffoli gates, but stuck-at faults on bit lines would imply the loss of reversibility.

Physical implementation of reversible circuits using switching devices with limited or no gain may generate new applications. Aside from quantum circuits, interesting examples include implementations in CMOS powered by circuit inputs [Desoete and De Vos 2002; De Vos 2010a; Skoneczny et al. 2008] as well as photonic circuits [Politi et al. 2009; Gao et al. 2010].

Design and verification tools for reversible and quantum circuits have been developed and reported by a number of groups, but in most cases they are point tools built to demonstrate specific algorithms. In contrast, conventional circuit-design environments employ long chains of inter-operating software tools. Such powerful software may be necessary to scale reversible and quantum circuit design beyond its current limitations [Svore et al. 2006; Wille and Drechsler 2010]. On the other hand, there is danger of developing CAD tools that are not fully motivated by applications.

ACKNOWLEDGMENTS

We thank Dr. Dmitri Maslov, Dr. Vivek Shende, Héctor J. García, Prof. John P. Hayes, and Dr. Smita Krishnaswamy for proofreading early versions of the manuscript and for helpful discussions.

REFERENCES

AARONSON, S. AND GOTTESMAN, D. 2004. Improved simulation of stabilizer circuits. *Phys. Rev. A 70*, 052328, arXiv:quant–ph/0406196v5.

ALTEPETER, J., JEFFREY, E., AND KWIAT, P. 2005. Photonic state tomography. Advances in Atomic, Molecular, and Optical Physics Series, vol. 52. Academic Press, 105–159.

ARABZADEH, M. AND SAEEDI, M. 2011. RCviewer+, A viewer/analyzer for reversible and quantum circuits, version 1.88. *http://ceit.aut.ac.ir/QDA/RCV.htm*.

ARABZADEH, M., SAEEDI, M., AND SAHEB ZAMANI, M. 2010. Rule-based optimization of reversible circuits. *Asia and South Pacific Design Autom. Conf.*, 849–854.

ASANO, M. AND ISHII, C. 2005. New structural quantum circuit simulating a Toffoli gate. *arXiv:quant-ph/0512016*.

BACON, D. AND VAN DAM, W. 2010. Recent progress in quantum algorithms. *Commun. ACM 53*, 84–93.

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

BECKMAN, D., CHARI, A. N., DEVABHAKTUNI, S., AND PRESKILL, J. 1996. Efficient networks for quantum factoring. *Phys. Rev. A 54,* 2, 1034–1063.

BENNETT, C. H. 1973. Logical reversibility of computation. *IBM J. Resear. Deve. 17,* 6, 525–532.

BERGHOLM, V., VARTIAINEN, J. J., MÖTTÖNEN, M., AND SALOMAA, M. M. 2005. Quantum circuits with uniformly controlled one-qubit gates. *Phys. Rev. A 71*, 052330.

BOLLIG, B., LÖBBING, M., SAUERHOFF, M., AND WEGENER, I. 1999. On The Complexity of the Hidden Weighted Bit Function for Various BDD Models. *Informatique Theorique et Applications 33,* 2, 103–116.

BRYANT, R. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput. 35,* 8, 677–691.

BUSHNELL, M. AND AGRAWAL, V. 2000. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Kluwer, USA.

CHEUNG, D. 2004. Improved bounds for the approximate QFT. *Int'l Symp. on Inf. and Communication Technologies*, 1–6.

CHEUNG, D., MASLOV, D., MATHEW, J., AND PRADHAN, D. K. 2009. On the design and optimization of a quantum polynomial-time attack on elliptic curve cryptography. *Quant. Inf. Comput. 9,* 7&8, 610–621.

CHEUNG, D., MASLOV, D., AND SEVERINI., S. 2007. Translation techniques between quantum circuit architectures. *Workshop on Quant. Inf. Proc.*.

CHILDS, A. M. AND VAN DAM, W. 2010. Quantum algorithms for algebraic problems. *Rev. Mod. Phys. 82,* 1, 1–52.

CHOI, B. AND VAN METER, R. 2008. Effects of interaction distance on quantum addition circuits. *quant-ph/0809.4317*.

CUCCARO, S. A., DRAPER, T. G., KUTIN, S. A., AND MOULTON, D. P. 2005. A new quantum ripple-carry addition circuit. *Workshop on Quant. Inf. Proc.*.

DALLY, W. AND TOWLES, B. 2003. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., USA.

DE VOS, A. 2010a. Reversible computer hardware. *Electron. Notes Theor. Comput. Sci. 253,* 6, 17–22.

DE VOS, A. 2010b. *Reversible Computing*. WileyVCH.

DE VOS, A., RAA, B., AND STORME, L. 2002. Generating the group of reversible logic gates. *J. of Phys. A 35,* 33, 7063.

DESOETE, B. AND DE VOS, A. 2002. A reversible carry-look-ahead adder using control gates. *Integr. VLSI J. 33,* 1, 89–104.

DIMITROV, V. S., JARVINEN, K. U., AND ADIKARI, J. 2011. Area-efficient multipliers based on multiple-radix representations. *IEEE Trans. Comput. 60,* 2, 189–201.

DONALD, J. AND JHA, N. K. 2008. Reversible logic synthesis with Fredkin and Peres gates. *J. Emerg. Technol. Comput. Sys. 4,* 1, 2:1–2:19.

DOUÇOT, B., IOFFE, L. B., AND VIDAL, J. 2004. Discrete non-Abelian gauge theories in Josephson-junction arrays and quantum computation. *Phys. Rev. B 69,* 21, 214501.

EGNER, S., PÜSCHEL, M., AND BETH, T. 1997. Decomposing a permutation into a conjugated tensor product. *Int'l Symp. on Symbolic and Algebraic Computation*, 101–108.

FAZEL, K., THORNTON, M., AND RICE, J. 2007. ESOP-based Toffoli Gate Cascade Generation. *Proceedings of the IEEE Pacific Rim Conference on Communications*, 206–209.

FEI, X., JIANG-FENG, D., MING-JUN, S., XIAN-YI, Z., RONG-DIAN, H., AND JI-HUI, W. 2002. Realization of the Fredkin gate by three transition pulses in a nuclear magnetic resonance quantum information processor. *Chinese Phys. Lett. 19,* 8, 1048.

FEYNMAN, R. 1986. Quantum mechanical computers. *Found. Phys. 16,* 6, 507–531.

FOWLER, A. G., DEVITT, S. J., AND HOLLENBERG, L. C. L. 2004a. Implementation of Shor's algorithm on a linear nearest neighbour qubit array. *Quant. Inf. Comput. 4*, 237–245.

FOWLER, A. G., HILL, C. D., AND HOLLENBERG, L. C. L. 2004b. Quantum error correction on linear nearest neighbor qubit arrays. *Phys. Rev. A 69,* 4, 042314.1–042314.4.

FREDKIN, E. F. AND TOFFOLI, T. 1982. Conservative logic. *Int. J. Theor. Phys. 21,* 3/4, 219–253.

GAO, W.-B., XU, P., YAO, X.-C., GÜHNE, O., CABELLO, A., LU, C.-Y., PENG, C.-Z., CHEN, Z.-B., AND PAN, J.-W. 2010. Experimental realization of a controlled-NOT gate with four-photon six-qubit cluster states. *Phys. Rev. Lett. 104,* 2, 020501.

GLÜCK, R. AND KAWABE, M. 2005. A method for automatic program inversion based on lr(0) parsing. *Fundam. Inf. 66,* 4, 367–395.

GOLUBITSKY, O., FALCONER, S. M., AND MASLOV, D. 2010. Synthesis of the optimal 4-bit reversible circuits. *Design Autom. Conf.*, 653–656.

GOLUBITSKY, O. AND MASLOV, D. 2011. A study of optimal 4-bit reversible toffoli circuits and their synthesis. *arXiv:1103.2686*.

GRASSL, M. 2003. Circuits for quantum error-correcting codes. *http://iaks-www.ira.uka.de/home/grassl/QECC/index.html*.

GROSSE, D., WILLE, R., DUECK, G., AND DRECHSLER, R. 2009. Exact multiple-control Toffoli network synthesis with SAT techniques. *IEEE Trans. CAD 28,* 5, 703–715.

GUPTA, P., AGRAWAL, A., AND JHA, N. 2006. An algorithm for synthesis of reversible logic circuits. *IEEE Trans. CAD 25,* 11, 2317–2330.

HACHTEL, G. D. AND SOMENZI, F. 2000. *Logic Synthesis and Verification Algorithms*. Kluwer.

HÄFFNER, H., HÄNSEL, W., ROOS, C. F., BENHELM, J., AL KAR, D. C., 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.

HILEWITZ, Y. AND LEE, R. B. 2008. Fast bit gather, bit scatter and bit permutation instructions for commodity microprocessors. *J. of Signal Proc. Sys. 53*, 145–169.

HIRATA, Y., NAKANISHI, M., YAMASHITA, S., AND NAKASHIMA, Y. 2011. An efficient conversion of quantum circuits to a linear nearest neighbor architecture. *Quant. Inf. Comput. 11,* 1–2, 0142–0166.

HUNG, W., SONG, X., YANG, G., YANG, J., AND PERKOWSKI, M. 2006. Optimal synthesis of multiple output Boolean functions using a set of quantum gates by symbolic reachability analysis. *IEEE Trans. CAD 25,* 9, 1652–1663.

IWAMA, K., KAMBAYASHI, Y., AND YAMASHITA, S. 2002. Transformation rules for designing CNOT-based quantum circuits. *Design Autom. Conf.*, 419–424.

JERRUM, M., SINCLAIR, A., AND VIGODA, E. 2004. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM 51,* 4, 671–697.

KANE, B. 1998. A silicon-based nuclear spin quantum computer. *Nature 393*, 133–137.

KERNTOPF, P. 2004. A new heuristic algorithm for reversible logic synthesis. *Design Autom. Conf.*, 834–837.

KIM, S., ZIESLER, C. H., AND PAPAEFTHYMIOU, M. C. 2005. Charge-recovery computing on silicon. *IEEE Trans. Comput. 54,* 6, 651–659.

KORF, R. 1999. *Artificial intelligence search algorithms*. in Algorithms Theory Computation Handbook, CRC Press.

KUTIN, S., MOULTON, D., AND SMITHLINE, L. 2007. Computation at a distance. *Chicago J. of Theor. Comput. Sci.*.

KUTIN, S. A. 2007. Shor's algorithm on a nearest-neighbor machine. *Asian Conf. on Quant. Inf. Sci.*.

LAFOREST, M., SIMON, D., BOILEAU, J.-C., BAUGH, J., DITTY, M., AND LAFLAMME, R. 2007. Using error correction to determine the noise model. *Phys. Rev. A 75,* 1, 133–137.

LAGARIAS, J. C. AND ODLYZKO, A. M. 1987. Computing [pi](x): An analytic method. *J. of Algorithms 8,* 2, 173–191.

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

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

MARKOV, I. L. AND ROY, J. A. 2003. On sub-optimality and scalability of logic synthesis tools. *Int'l Workshop on Logic Synth.*.

MARKOV, I. L. AND SAEEDI, M. 2012. Constant-optimized quantum circuits for modular multiplication and exponentiation. *Quantum Info. Comput. 12,* 5-6, 361–394.

MARKOV, I. L. AND SAEEDI, M. 2013. Faster quantum number factoring via circuit synthesis. *Phys. Rev. A 87*, 012310.

MASLOV, D. 2007. Linear depth stabilizer and quantum Fourier transformation circuits with no auxiliary qubits in finite neighbor quantum architectures. *Phys. Rev. A 76*.

MASLOV, D. Feb 2011. Reversible logic synthesis benchmarks page. *http://www.cs.uvic.ca/~dmaslov/*.

MASLOV, D. AND DUECK, G. 2003. Improved quantum cost for n-bit Toffoli gates. *Electronics Letters 39,* 25, 1790–1791.

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

MASLOV, D., DUECK, G., AND MILLER, D. 2005a. Synthesis of Fredkin-Toffoli reversible networks. *IEEE Trans. VLSI 13,* 6, 765–769.

MASLOV, D., DUECK, G., AND MILLER, D. 2005b. Toffoli network synthesis with templates. *IEEE Trans. CAD 24,* 6, 807–817.

MASLOV, D., DUECK, G., MILLER, D., AND NEGREVERGNE, C. 2008a. Quantum circuit simplification and level compaction. *IEEE Trans. CAD 27,* 3, 436–444.

MASLOV, D., DUECK, G. W., AND MILLER, D. M. 2007. Techniques for the synthesis of reversible Toffoli networks. *ACM Trans. Des. Autom. Electron. Sys. 12,* 4, 42:1–42:28.

MASLOV, D., FALCONER, S. M., AND MOSCA, M. 2008b. Quantum circuit placement. *IEEE Trans. CAD 27,* 4, 752–763.

MASLOV, D. AND SAEEDI, M. 2011. Reversible circuit optimization via leaving the Boolean domain. *IEEE Trans. CAD 30,* 6, 806 – 816.

MCGREGOR, J. P. AND LEE, R. B. 2003. Architectural techniques for accelerating subword permutations with repetitions. *IEEE Trans. VLSI 11*, 325–335.

MILLER, D. AND SASANIAN, Z. 2010. Improving the NCV realization of multiple-control Toffoli gates. *Int'l Workshop on Boolean Problems*, 37–44.

MILLER, D. M., MASLOV, D., AND DUECK, G. W. 2003. A transformation based algorithm for reversible logic synthesis. *Design Autom. Conf.*, 318–323.

MILLER, D. M., WILLE, R., AND DRECHSLER, R. 2010. Reducing reversible circuit cost by adding lines. *Int'l Symp. on Multiple-Valued Logic*, 217–222.

MISHCHENKO, A. AND PERKOWSKI, M. 2002. Logic synthesis of reversible wave cascades. *Int'l Workshop on Logic Synth.*, 197–202.

MOORE, C. AND CRUTCHFIELD, J. P. 2000. Quantum automata and quantum grammars. *Theor. Comput. Sci. 237,* 1-2, 275–306.

MORITA, K. 2008. Reversible computing and cellular automata - a survey. *Theor. Comput. Sci. 395,* 1, 101–131.

MÖTTÖNEN, M. AND VARTIAINEN, J. J. 2006. *Decompositions of general quantum gates*. Ch. 7 in Trends in Quantum Computing Research, NOVA Publishers.

NEGREVERGNE, C., MAHESH, T. S., RYAN, C. A., DITTY, M., CYR-RACINE, F., POWER, W., BOULANT, N., HAVEL, T., CORY, D. G., AND LAFLAMME, R. 2006. Benchmarking quantum control methods on a 12-qubit system. *Phys. Rev. Lett. 96,* 17.

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

PATEL, K. N., MARKOV, I. L., AND HAYES, J. P. 2008. Optimal synthesis of linear reversible circuits. *Quant. Inf. Comput. 8,* 3-4, 282–294.

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

PÉREZ-DELGADO, C. A., MOSCA, M., CAPPELLARO, P., AND CORY, D. G. 2006. Single spin measurement using cellular automata techniques. *Phys. Rev. Lett. 97,* 10, 100501.

POLIAN, I., FIEHN, T., BECKER, B., AND HAYES, J. P. 2005. A family of logical fault models for reversible circuits. *Asian Test Symp.*, 422–427.

POLITI, A., MATTHEWS, J. C. F., AND O'BRIEN, J. L. 2009. Shor's quantum factoring algorithm on a photonic chip. *Science 325,* 5945, 1221.

PRASAD, A. K., SHENDE, V. V., MARKOV, I. L., HAYES, J. P., AND PATEL, K. N. 2006. Data structures and algorithms for simplifying reversible circuits. *J. Emerg. Technol. Comput. Sys. 2,* 4, 277–293.

RENTERGEM, Y. V., VOS, A. D., AND KEYSER, K. D. 2007. Six synthesis methods for reversible logic. *Open Sys. & Inf. Dynamics 14,* 1, 91–116.

SAEEDI, M., ARABZADEH, M., SAHEB ZAMANI, M., AND SEDIGHI, M. 2011a. Block-based quantum-logic synthesis. *Quant. Inf. Comput. 11,* 3-4, 0262–0277.

SAEEDI, M., SAHEB ZAMANI, M., AND SEDIGHI, M. 2007a. On the behavior of substitution-based reversible circuit synthesis algorithms: investigation and improvement. *Int'l Symp. on VLSI*, 428–436.

SAEEDI, M., SAHEB ZAMANI, M., SEDIGHI, M., AND SASANIAN, Z. 2010a. Reversible circuit synthesis using a cycle-based approach. *J. Emerg. Technol. Comput. Sys. 6,* 4, 13:1–13:26.

SAEEDI, M., SEDIGHI, M., AND SAHEB ZAMANI, M. 2007b. A novel synthesis algorithm for reversible circuits. *Int'l Conf. on Computer-Aided Design*, 65–68.

SAEEDI, M., SEDIGHI, M., AND SAHEB ZAMANI, M. 2010b. A library-based synthesis methodology for reversible logic. *Microelectron. J. 41,* 4, 185–194.

SAEEDI, M., WILLE, R., AND DRECHSLER, R. 2011b. Synthesis of quantum circuits for linear nearest neighbor architectures. *Quant. Inf. Proc. 10,* 3, 355–377.

SHENDE, V., BULLOCK, S., AND MARKOV, I. L. 2006. Synthesis of quantum-logic circuits. *IEEE Trans. CAD 25,* 6, 1000–1010.

SHENDE, V. V. AND MARKOV, I. L. 2009. On the CNOT-cost of TOFFOLI gates. *Quant. Inf. Comput. 9,* 5-6, 461–486.

SHENDE, V. V., MARKOV, I. L., AND BULLOCK, S. S. 2004. Minimal universal two-qubit quantum circuits. *Phys. Rev. A 69*, 062321.

SHENDE, V. V., PRASAD, A. K., MARKOV, I. L., AND HAYES, J. P. 2003. Synthesis of reversible logic circuits. *IEEE Trans. CAD 22,* 6, 710–722.

SHI, Y.-Y., DUAN, L.-M., AND VIDAL, G. 2006. Classical simulation of quantum many-body systems with a tree tensor network. *Phys. Rev. A 74,* 2, 022320.

SHI, Z. AND LEE, R. B. 2000. Bit permutation instructions for accelerating software cryptography. *Int'l Conf. on Applic.-Spec. Sys., Architectures, and Processors*, 138–148.

SKINNER, A. J., DAVENPORT, M. E., AND KANE, B. E. 2003. Hydrogenic spin quantum computing in silicon: A digital approach. *Phys. Rev. Lett. 90,* 8, 087901.

SKONECZNY, M., VAN RENTERGEM, Y., AND DE VOS, A. 2008. Reversible fourier transform chip. *Mixed Design of Integrated Circuits and Sys.*, 281–286.

SOEKEN, M., FREHSE, S., WILLE, R., AND DRECHSLER, R. 2010a. RevKit: A toolkit for reversible circuit design. *Workshop on Reversible Computation*. RevKit is available at http://www.revkit.org.

SOEKEN, M., WILLE, R., DUECK, G. W., AND DRECHSLER, R. 2010b. Window optimization of reversible and quantum circuits. *Design & Diagnostics of Elec. Circ. & Sys.*, 341–345.

STORME, L., VOS, A. D., AND JACOBS, G. 1999. Group theoretical aspects of reversible logic gates. *J. of Universal Comput. Sci. 5,* 5, 307–321.

SVORE, K. M., AHO, A. V., CROSS, A. W., CHUANG, I., AND MARKOV, I. L. 2006. A layered software architecture for quantum computing design tools. *Computer 39*, 74–83.

TAKAHASHI, Y. AND KUNIHIRO, N. 2008. A fast quantum circuit for addition with few qubits. *Quant. Inf. Comput. 8,* 6-7, 636–649.

TAKAHASHI, Y., KUNIHIRO, N., AND OHTA, K. 2007. The quantum Fourier transform on a linear nearest neighbor architecture. *Quant. Inf. Comput. 7,* 4, 383–391.

TAKAHASHI, Y., TANI, S., AND KUNIHIRO, N. 2010. Quantum addition circuits and unbounded fan-out. *Quant. Inf. Comput. 10,* 9&10, 872–890.

TOFFOLI, T. 1980. Reversible computing. Springer, 632. Technical Memo MIT/LCS/TM-151, MIT Lab. for Comput. Sci.

VAN METER, R. AND ITOH, K. M. 2005. Fast quantum modular exponentiation. *Phys. Rev. A 71,* 5, 052320.

VAN METER, R. AND OSKIN, M. 2006. Architectural implications of quantum computing technologies. *J. Emerg. Technol. Comput. Sys. 2,* 1, 31–63.

VIAMONTES, G., MARKOV, I. L., AND HAYES, J. P. 2009. *Quantum Circuit Simulation*. Springer.

VIAMONTES, G. F., MARKOV, I. L., AND HAYES, J. P. 2007. Checking equivalence of quantum circuits and states. *Int'l Conf. on Computer-Aided Design*, 69–74.

VISAN, A. M., POLYAKOV, A., SOLANKI, P. S., ARYA, K., DENNISTON, T., AND COOPERMAN, G. 2009. Temporal debugging using URDB. *CoRR abs/0910.5046*.

VON NEUMANN, J. 1966. *Theory of Self-Reproducing Automata*. Univ. of Illinois Press, USA.

WILLE, R. AND DRECHSLER, R. 2009. BDD-based synthesis of reversible logic for large functions. *Design Autom. Conf.*, 270–275.

WILLE, R. AND DRECHSLER, R. 2010. *Towards a Design Flow for Reversible Logic*. Springer.

WILLE, R., GROSSE, D., MILLER, D. M., AND DRECHSLER, R. 2009. Equivalence checking of reversible circuits. *Int'l Symp. on Multiple-Valued Logic*, 324–330.

WILLE, R., GROSSE, D., TEUBER, L., DUECK, G. W., AND DRECHSLER, R. 2008a. RevLib: An online resource for reversible functions and reversible circuits. *Int'l Symp. on Multiple-Valued Logic*, 220–225.

WILLE, R., LE, H. M., DUECK, G. W., AND GROSSE, D. 2008b. Quantified synthesis of reversible logic. *Design, Autom., and Test Eur.*, 1015–1020.

WILLE, R., OFFERMANN, S., AND DRECHSLER, R. 2010a. SyReC: A Programming Language for Synthesis of Reversible Circuits. In *Forum on specification & Design Languages*.

WILLE, R., SOEKEN, M., AND DRECHSLER, R. 2010b. Reducing the number of lines in reversible circuits. *Design Autom. Conf.*, 647–652.

YAMASHITA, S. AND MARKOV, I. L. 2010. Fast equivalence-checking for quantum circuits. *Quant. Inf. Comput. 9,* 9-10, 721–734.

YANG, G., SONG, X., HUNG, W. N., XIE, F., AND PERKOWSKI, M. A. 2006. Group theory based synthesis of binary reversible circuits. *Lec. Notes in Comp. Sci. 3959/2006*, 365–374.

YANG, G., SONG, X., HUNG, W. N. N., AND PERKOWSKI, M. A. 2008. Bi-directional synthesis of 4-bit reversible circuits. *Comput. J. 51*, 207–215.

YOKOYAMA, T., AXELSEN, H. B., AND GLÜCK, R. 2008. Principles of a reversible programming language. *Comput. Frontiers*, 43–54.

ZHANG, J., VALA, J., SASTRY, S., AND WHALEY, K. B. 2003. Geometric theory of nonlocal two-qubit operations. *Phys. Rev. A 67,* 4, 042313.