PHYSICAL REVIEW A VOLUME 52, NUMBER 5 NOVEMBER 1995

Elementary gates for quantum computation

Adriano Barenco,[1,*] Charles H. Bennett,[2,†] Richard Cleve,[3,‡] David P. DiVincenzo,[2,†] Norman Margolus,[4,§] Peter Shor,[5,||] Tycho Sleator,[6,¶] John A. Smolin,[7,**] and Harald Weinfurter[8,††]

[1]*Clarendon Laboratory, Oxford University, Oxford OX1 3PU, United Kingdom*
[2]*IBM Research, Yorktown Heights, New York 10598*
[3]*Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4*
[4]*Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139*
[5]*AT&T Bell Laboratories, Murray Hill, New Jersey 07974*
[6]*Physics Department, New York University, New York, New York 10003*
[7]*Physics Department, University of California at Los Angeles, Los Angeles, California 90024*
[8]*Institute for Experimental Physics, University of Innsbruck, A-6020 Innsbruck, Austria*

(Received 22 March 1995)

We show that a set of gates that consists of all one-bit quantum gates [U(2)] and the two-bit exclusive-OR gate [that maps Boolean values (x,y) to $(x,x\oplus y)$] is universal in the sense that all unitary operations on arbitrarily many bits n [U(2^n)] can be expressed as compositions of these gates. We investigate the number of the above gates required to implement other gates, such as generalized Deutsch-Toffoli gates, that apply a specific U(2) transformation to one input bit if and only if the logical AND of all remaining input bits is satisfied. These gates play a central role in many proposed constructions of quantum computational networks. We derive upper and lower bounds on the exact number of elementary gates required to build up a variety of two- and three-bit quantum gates, the asymptotic number required for n-bit Deutsch-Toffoli gates, and make some observations about the number required for arbitrary n-bit unitary operations.

PACS number(s): 03.65.Ca, 07.05.Bx, 02.70.Rw, 89.80.+h

I. BACKGROUND

It has recently been recognized, after 50 years of using the paradigms of classical physics (as embodied in the Turing machine) to build a theory of computation, that quantum physics provides another paradigm with clearly different and possibly much more powerful features than established computation theory. In quantum computation, the state of the computer is described by a state vector Ψ, which is a complex linear superposition of all binary states of the bits $x_m \in\{0,1\}$:

$$\Psi(t)=\sum_{x\in\{0,1\}^m}\alpha_x|x_1,\ldots,x_m\rangle,\qquad \sum_x|\alpha_x|^2=1\ .$$

The state's evolution in the course of time t is described by a *unitary* operator U on this vector space, i.e., a linear transformation that is bijective and length preserving. This unitary evolution on a normalized state vector is known to be the correct physical description of an isolated system evolving in time according to the laws of quantum mechanics [1].

Historically, the idea that the quantum mechanics of isolated systems should be studied as a new formal system for computation arose from the recognition 20 years ago that computation could be made reversible within the paradigm of classical physics. It is possible to perform any computation in a way that is reversible both *logically*, i.e., the computation is a sequence of bijective transformations, and *thermodynamically*, i.e., the computation could in principle be performed by a physical apparatus dissipating arbitrarily little energy [2]. A formalism for constructing reversible Turing machines and reversible gate arrays (i.e., reversible combinational logic) was developed. Fredkin and Toffoli [3] showed that there exists a three-bit "universal gate" for reversible computation, that is, a gate that, when applied in succession to different triplets of bits in a gate array, could be used to simulate any arbitrary reversible computation. (Two-bit gates such as NAND that are universal for ordinary computation are not reversible.) Toffoli's version [4] of the universal reversible gate will figure prominently in the body of this paper.

Quantum physics is also reversible because the reverse-time evolution specified by the unitary operator $U^{-1}=U^\dagger$ always exists; as a consequence, several workers recognized that reversible computation could be executed within a quantum-mechanical system. Quantum-mechanical Turing machines [5,6], gate arrays [7], and cellular automata [8] have been discussed and physical realizations of Toffoli's [9–11] and Fredkin's [12–14] universal three-bit gates within various quantum-mechanical physical systems have been proposed.

[*]Electronic address: a.barenco@mildred.physics.ox.ac.uk
[†]Electronic address: bennetc/divince@watson.ibm.com
[‡]Electronic address: cleve@cpsc.ucalgary.ca
[§]Electronic address: nhm@im.lcs.mit.edu
[||]Electronic address: shor@research.att.com
[¶]Electronic address: tycho@sleator.physics.nyu.edu
[**]Electronic address: smolin@vesta.physics.ucla.edu
[††]Electronic address: harald.weinfurter@uibk.ac.at

While reversible computation is contained within quantum mechanics, it is a small subset: the time evolution of a classical reversible computer is described by unitary operators whose matrix elements are only zero or one, arbitrary complex numbers are not allowed. Unitary time evolution can of course be *simulated* by a classical computer (e.g., an analog optical computer governed by Maxwell's equations [15]), but the dimension of the unitary operator thus attainable is bounded by the number of classical degrees of freedom, i.e., roughly proportional to the size of the apparatus. In contrast, a quantum computer with m physical bits (see the definition of the state above) can perform unitary operations in a space of 2^m dimensions, exponentially larger than its physical size.

Deutsch [16] introduced a quantum Turing machine intended to generate and operate on arbitrary superpositions of states and proposed that, aside from simulating the evolution of quantum systems more economically than known classical methods, it might also be able to solve certain *classical* problems, i.e., problems with a classical input and output, faster than on any classical Turing machine. In a series of artificial settings, with appropriately chosen oracles, quantum computers were shown to be qualitatively stronger than classical ones [17–20], culminating in Shor's [21,22] discovery of quantum polynomial-time algorithms for two important natural problems, viz., factoring and discrete logarithm, for which no polynomial-time classical algorithm was known. The search for other such problems and the physical question of the feasibility of building a quantum computer are major topics of investigation today [23].

The formalism we use for quantum computation, which we call a quantum "gate array," was introduced by Deutsch [24], who showed that a simple generalization of the Toffoli gate [the three-bit gate $\wedge_2(R_x)$, in the language introduced later in this paper] suffices as a universal gate for quantum computing. The quantum gate array is the natural quantum generalization of acyclic combinational logic "circuits" studied in conventional computational complexity theory. It consists of quantum "gates," interconnected without fanout or feedback by quantum "wires." The gates have the same number of inputs as outputs and a gate of n inputs carries a unitary operation of the group $U(2^n)$, i.e., a generalized rotation in a Hilbert space of dimension 2^n. Each wire represents a quantum bit, or *qubit* [25,26], i.e., a quantum system with a two-dimensional Hilbert space, capable of existing in a superposition of Boolean states and of being entangled with the states of other qubits. Where there is no danger of confusion, we will use the term "bit" in either the classical or quantum sense. Just as classical bit strings can represent the discrete states of arbitrary finite dimensionality, so a string of n qubits can be used to represent quantum states in any Hilbert space of dimensionality up to 2^n. The analysis of quantum Turing machines [20] is complicated by the fact that not only the data but also the control variables, e.g., head position, can exist in a superposition of classical states. Fortunately, Yao has shown [27] that acyclic quantum gate arrays can simulate quantum Turing machines. Gate arrays are easier to think about since the control variables, i.e., the wiring diagram itself and the number of steps of computation executed so far, can be thought of as classical with only the data in the wires being quantum.

Here we derive a series of results that provide tools for the building up of unitary transformations from simple gates. We build on other recent results that simplify and extend Deutsch's original discovery [24] of a three-bit universal quantum logic gate. As a consequence of the greater power of quantum computing as a formal system, there are many more choices for the universal gate than in classical reversible computing. In particular, DiVincenzo [28] showed that two-bit universal quantum gates are also possible; Barenco [29] extended this to show than almost any two-bit gate (within a certain restricted class) is universal and Lloyd [30] and Deutsch [31] have shown that in fact almost any two-bit or n-bit ($n \geqslant 2$) gate is also universal. A closely related construction for the Fredkin gate has been given [32]. In the present paper we take a somewhat different tack, showing that a nonuniversal, classical two-bit gate, in conjunction with quantum one-bit gates, is also universal; we believe that the present work, along with the preceding ones, covers the full range of possible repertoires for quantum gate array construction.

With our universal-gate repertoire, we also exhibit a number of efficient schemes for building up certain classes of n-bit operations with these gates. A variety of strategies for constructing gate arrays efficiently will surely be very important for understanding the full power of quantum mechanics for computation; the construction of such efficient schemes has already proved very useful for understanding the scaling of Shor's prime factorization [33]. In the present work we in part build upon the strategy introduced by Sleator and Weinfurter [9], who exhibited a scheme for obtaining the Toffoli gate with a sequence of exactly five two-bit gates. We find that their approach can be generalized and extended in a number of ways to obtain more general, efficient gate constructions. Some of the results presented here have no obvious connection with previous gate-assembly schemes.

We will not touch at all on the great difficulties attendant on the actual physical realization of a quantum computer; the problems of error correction [34] and quantum coherence [35,36] are very serious ones. We refer the reader to [37] for a comprehensive discussion of these difficulties.

II. INTRODUCTION

We begin by introducing some basic ideas and notation. For any unitary matrix

$$U=\begin{pmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{pmatrix}$$

and $m \in \{0,1,2,\ldots\}$, define the $(m+1)$-bit ($2^{(m+1)}$-dimensional) operator $\wedge_m(U)$ as

$$\wedge_m(U)(|x_1,\ldots,x_m,y\rangle) = \begin{cases} u_{y0}|x_1,\ldots x_m,0\rangle + u_{y1}|x_1,\ldots,x_n,1\rangle & \text{if } \wedge_{k=1}^m x_k = 1 \\ |x_1,\ldots,x_m,y\rangle & \text{if } \wedge_{k=1}^m x_k = 0 \end{cases}$$

for all $x_1,\ldots,x_m,y \in \{0,1\}$. (In more ordinary language, $\wedge_{k=1}^m x_k$ denotes the AND of the Boolean variables $\{x_k\}$.)

Note that $\wedge_0(U)$ is equated with U. The $2^{(m+1)}\times 2^{(m+1)}$ matrix corresponding to $\wedge_m(U)$ is

$$\begin{pmatrix} 1 & & & & & \\ & 1 & & & & \\ & & \ddots & & & \\ & & & 1 & & \\ & & & & u_{00} & u_{01} \\ & & & & u_{10} & u_{11} \end{pmatrix}$$

(where the basis states are lexicographically ordered, i.e., $|000\rangle$, $|001\rangle, \ldots, |111\rangle$).

When

$$U=\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

$\wedge_m(U)$ is the so-called Toffoli gate [4] with $m+1$ input bits, which maps $|x_1, \ldots, x_m, y\rangle$ to $|x_1, \ldots, x_m, (\wedge_{k=1}^m x_k)\oplus y\rangle$. For a general U, $\wedge_m(U)$ can be regarded as a generalization of the Toffoli gate, which, on input $|x_1, \ldots, x_m, y\rangle$, applies U to y if and only if $\wedge_{k=1}^m x_k = 1$.

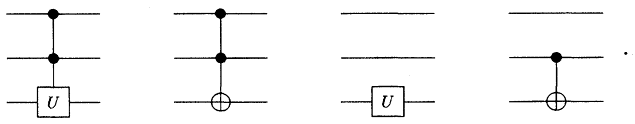
As shown by one of us [31,29], "almost any" single $\wedge_1(U)$ gate is universal in the sense that by successive application of this gate to pairs of bits in an n-bit network, any unitary transformation may be approximated with arbitrary accuracy. (It suffices for U to be specified by Euler angles that are not a rational multiple of π.)

We show that in some sense this result can be made even simpler in that any unitary transformation in a network can always be constructed out of only the "classical" two-bit gate $\wedge_1(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix})$ along with a set of one-bit operations [of the form $\wedge_0(U)$]. This is a remarkable result from the perspective of classical reversible computation because it is well known that the classical analog of this assertion, which is that all invertible Boolean functions can be implemented with $\wedge_1(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix})$ and $\wedge_0(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix})$ gates [38], is false. In fact, it is well known that only a tiny fraction of Boolean functions (those that are linear with respect to mod2 arithmetic) can be generated with these gates [39].

We will also exhibit a number of explicit constructions of $\wedge_m(U)$ using $\wedge_1(U)$, which can all be made polynomial in m. It is well known [4] that the analogous family of constructions in classical reversible logic that involve building $\wedge_m(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix})$ from the three-bit Toffoli gate $\wedge_2(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix})$ is also polynomial in m. We will exhibit one important difference between the classical and the quantum constructions, however; Toffoli showed [4] that the classical $\wedge_m$'s could not be built without the presence of "work bits" to store intermediate results of the calculation. By contrast, we show that the quantum logic gates can always be constructed with the use of *no* workspace whatsoever. Similar computations in the classical setting (that use very few or no work bits) appeared in the work of Cleve [40] and Ben-Or and Cleve [41]. Still, the presence of a workspace plays an important role in the quantum gate constructions: we find that to implement a family of $\wedge_m$ gates exactly, the time required for our implementation can be reduced from $\Theta(m^2)$ to $\Theta(m)$ merely by the introduction of *one* bit for workspace.

III. NOTATION

We adopt a version of Feynman's [7] notation to denote $\wedge_m(U)$ gates and Toffoli gates in quantum networks as follows:



The first network contains a $\wedge_2(U)$ gate and the second one contains a three-bit Toffoli gate [42]. The third and fourth networks contain a $\wedge_0(U)$ and a two-bit reversible exclusive-or (simply called XOR henceforth) gate, respectively. The XOR gate is introduced as the "measurement gate" in [24] and will play a very prominent role in many of the constructions we describe below. Throughout this paper, when we refer to a *basic* operation, we mean either a $\wedge_0(U)$ gate or this two-bit XOR gate.

In all the gate-array diagrams shown in this paper, we use the usual convention that time advances from left to right, so that the leftmost gate operates first, etc.

IV. MATRIX PROPERTIES

Lemma 4.1. Every unitary 2×2 matrix can be expressed as

$$\begin{pmatrix} e^{i\delta} & 0 \\ 0 & e^{i\delta} \end{pmatrix}\begin{pmatrix} e^{i\alpha/2} & 0 \\ 0 & e^{-i\alpha/2} \end{pmatrix}\begin{pmatrix} \cos\theta/2 & \sin\theta/2 \\ -\sin\theta/2 & \cos\theta/2 \end{pmatrix} \times\begin{pmatrix} e^{i\beta/2} & 0 \\ 0 & e^{-i\beta/2} \end{pmatrix},$$

where δ, α, θ, and β are real valued. Moreover, any special unitary 2×2 matrix (i.e., with unity determinant) can be expressed as

$$\begin{pmatrix} e^{i\alpha/2} & 0 \\ 0 & e^{-i\alpha/2} \end{pmatrix}\begin{pmatrix} \cos\theta/2 & \sin\theta/2 \\ -\sin\theta/2 & \cos\theta/2 \end{pmatrix}\begin{pmatrix} e^{i\beta/2} & 0 \\ 0 & e^{-i\beta/2} \end{pmatrix}.$$

Proof. Since a matrix is unitary if and only if its row vectors and column vectors are orthonormal, every 2×2 unitary matrix is of the form

$$\begin{pmatrix} e^{i(\delta+\alpha/2+\beta/2)}\cos\theta/2 & e^{i(\delta+\alpha/2-\beta/2)}\sin\theta/2 \\ -e^{i(\delta-\alpha/2+\beta/2)}\sin\theta/2 & e^{i(\delta-\alpha/2-\beta/2)}\cos\theta/2 \end{pmatrix},$$

where δ, α, θ, and β are real valued. The first factorization above now follows immediately. In the case of special unitary matrices, the determinant of the first matrix must be 1, which implies $e^{i\delta}=\pm 1$, so the first matrix in the product can be absorbed into the second one. ■

Definition. In view of the above lemma, we define the following:

$$R_y(\theta)\begin{pmatrix} \cos\theta/2 & \sin\theta/2 \\ -\sin\theta/2 & \cos\theta/2 \end{pmatrix}$$

(a rotation by θ around $\hat{y}$ [43]) ,

$$R_z(\alpha)=\begin{pmatrix} e^{i\alpha/2} & 0 \\ 0 & e^{-i\alpha/2} \end{pmatrix}$$

(a rotation by α around $\hat{z}$) ,

$$\Phi(\delta)=\begin{pmatrix} e^{i\delta} & 0 \\ 0 & e^{i\delta} \end{pmatrix}$$

(a phase shift with respect to δ) ,

$$\sigma_x=\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$ (a "negation," one of the Pauli matrices) ,

$$I=\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$ (the identity matrix) .

Lemma 4.2. The following properties hold:

$$R_y(\theta_1)R_y(\theta_2)=R_y(\theta_1+\theta_2) ,$$

$$R_z(\alpha_1)R_z(\alpha_2)=R_z(\alpha_1+\alpha_2) ,$$

$$\Phi(\delta_1)\Phi(\delta_2)=\Phi(\delta_1+\delta_2) ,$$

$$\sigma_x\sigma_x=I ,$$

$$\sigma_xR_y(\theta)\sigma_x=R_y(-\theta) ,$$

$$\sigma_xR_z(\alpha)\sigma_x=R_z(-\alpha) .$$

Lemma 4.3. For any special unitary matrix W [$W \in \mathrm{SU}(2)$], there exist matrices A, B, and $C\in\mathrm{SU}(2)$ such that $ABC=I$ and $A\sigma_xB\sigma_xC=W$.

Proof. By Lemma 4.1, there exist α, θ, and β such that $W=R_z(\alpha)R_y(\theta)R_z(\beta)$. Set $A=R_z(\alpha)(\theta/2)$, $B=R_y(-\theta/2)R_z[-(\alpha+\beta)/2]$, and $C=R_z[(\beta-\alpha)/2]$. Then

$$ABC=R_z(\alpha)R_y\left(\frac{\theta}{2}\right)R_y\left(-\frac{\theta}{2}\right)R_z\left(-\frac{\alpha+\beta}{2}\right)R_z\left(\frac{\beta-\alpha}{2}\right)$$
$$=R_z(\alpha)R_z(-\alpha)=I$$

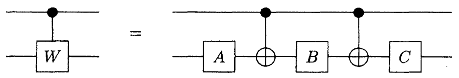
and

$$A\sigma_xB\sigma_xC=R_z(\alpha)R_y\left(\frac{\theta}{2}\right)\sigma_xR_y\left(-\frac{\theta}{2}\right)$$
$$\times R_z\left(-\frac{\alpha+\beta}{2}\right)\sigma_xR_z\left(\frac{\beta-\alpha}{2}\right)$$
$$=R_z(\alpha)R_y\left(\frac{\theta}{2}\right)\sigma_xR_y\left(-\frac{\theta}{2}\right)$$
$$\times \sigma_x\sigma_xR_z\left(-\frac{\alpha+\beta}{2}\right)\sigma_xR_z\left(\frac{\beta-\alpha}{2}\right)$$
$$=R_z(\alpha)R_y\left(\frac{\theta}{2}\right)R_y\left(\frac{\theta}{2}\right)R_z\left(\frac{\alpha+\beta}{2}\right)R_z\left(\frac{\beta-\alpha}{2}\right)$$
$$=R_z(\alpha)R_y(\theta)R_z(\beta)=W . \qquad \blacksquare$$

V. TWO-BIT NETWORKS

A. Simulation of general $\wedge_1(U)$ gates

Lemma 5.1. For a unitary 2×2 matrix W, a $\wedge_1(W)$ gate can be simulated by a network of the form

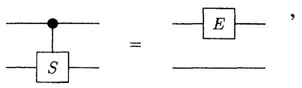


where A, B, and $C\in\mathrm{SU}(2)$ if and only if $W\in\mathrm{SU}(2)$.

Proof. For the "if" part, let A, B, and C be as in Lemma 4.3. If the value of the first (top) bit is 0, then $ABC=I$ is applied to the second bit. If the value of the first bit is 1, then $A\sigma_xB\sigma_xC=W$ is applied to the second bit.

For the "only if" part, note that $ABC=I$ must hold if the simulation is correct when the first bit is 0. Also, if the network simulates a $\wedge_1(W)$ gate, then $A\sigma_xB\sigma_xC=W$. Therefore, since $\det(A\sigma_xB\sigma_xC)=1$, W must also be special unitary. $\blacksquare$

Lemma 5.2. For any δ and $S=\Phi(\delta)$, a $\wedge_1(S)$ gate can be simulated by a network of the form



where E is unitary.

Proof. Let

$$E=R_z(-\delta)\Phi\left(\frac{\delta}{2}\right)=\begin{pmatrix} 1 & 0 \\ 0 & e^{i\delta} \end{pmatrix} .$$

Then the observation is that the 4×4 unitary matrix corresponding to each of the above networks is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{i\delta} & 0 \\ 0 & 0 & 0 & e^{i\delta} \end{pmatrix} . \qquad \blacksquare$$

Clearly, $\wedge_1(S)$ composed with $\wedge_1(W)$ yields $\wedge_1(SW)$. Thus, by noting that any unitary matrix U is of the form $U=SW$, where $S=\Phi(\delta)$ (for some δ) and $W\in\mathrm{SU}(2)$, we obtain the following.

Corollary 5.3. For any unitary 2×2 matrix U, a $\wedge_1(U)$ gate can be simulated by at most six basic gates: four one-bit gates $\wedge_0$ and two XOR gates $\wedge_1(\sigma_x)$.

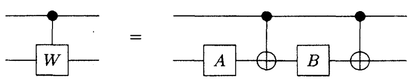
B. Special cases

In Sec. V A we have established a general simulation of a $\wedge_1(U)$ gate for an arbitrary unitary U. For special cases of U that may be of interest, a more efficient construction than that of Corollary 5.3 is possible. Clearly, Lemma 5.1 immediately yields a more efficient simulation for all special unitary matrices. For example, the "x-axis rotation matrix" [to use the language suggested by the mapping between SU(2) and SO(3), the group of rigid-body rotations [43]]

$$R_x(\theta)=\begin{pmatrix}\cos\theta/2 & i\sin\theta/2\\ i\sin\theta/2 & \cos\theta/2\end{pmatrix}=R_z\left(\frac{\pi}{2}\right)R_y(\theta)R_z\left(-\frac{\pi}{2}\right)$$

is special unitary. [R_x is of special interest because $\wedge_2(iR_x)$ is the "Deutsch gate" [24], which was shown to be universal for quantum logic.] For other specific SU(2) matrices an even more efficient simulation is possible.

Lemma 5.4. A $\wedge_1(W)$ gate can be simulated by a network of the form



where A and $B\in\mathrm{SU}(2)$ if and only if W is of the form

$$W=R_z(\alpha)R_y(\theta)R_z(\alpha)=\begin{pmatrix}e^{i\alpha}\cos\theta/2 & \sin\theta/2\\ -\sin\theta/2 & e^{-i\alpha}\cos\theta/2\end{pmatrix},$$

where α and θ are real valued.

Proof. For the "if" part, consider the simulation of $\wedge_1(W)$ that arises in Lemma 5.1 when $W=R_z(\alpha)R_y(\theta)R_z(\alpha)$. In this case, $A=R_z(\alpha)R_y(\theta/2)$, $B=R_y(-\theta/2)R_z(-\alpha)$, and $C=I$. Thus $B=A^\dagger$ and C can be omitted.

For the "only if" part, note that $B=A^\dagger$ must hold for the simulation to be valid when the first bit is 0. Therefore, if the first bit is 1, then $A\sigma_xA^\dagger\sigma_x$ is applied to the second bit. Now, the matrix $A\sigma_xA^\dagger$ has determinant -1 and is traceless (since its trace is the same as that of σ_x). By specializing the characterization of unitary matrices in Lemma 4.1 to traceless matrices with determinant -1, we conclude that $A\sigma_xA^\dagger$ must be of the form

$$A\sigma_xA^\dagger=\begin{pmatrix}\sin\theta/2 & e^{i\alpha}\cos\theta/2\\ e^{-i\alpha}\cos\theta/2 & -\sin\theta/2\end{pmatrix}.$$

Therefore,

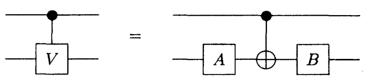
$$A\sigma_xA^\dagger\sigma_x=\begin{pmatrix}e^{i\alpha}\cos\theta/2 & \sin\theta/2\\ -\sin\theta/2 & e^{-i\alpha}\cos\theta/2\end{pmatrix},$$

as required. ■

Examples of matrices of the form of Lemma 5.4 are $R_y(\theta)$ itself, as well as $R_z(\alpha)=R_z(\alpha/2)R_y(0)R_z(\alpha/2)$. However, $R_x(\theta)$ is not of this form.

Finally, for certain U, we obtain an even greater simplification of the simulation of $\wedge_1(U)$ gates.

Lemma 5.5. A $\wedge_1(V)$ gate can be simulated by a construction of the form



where A and B are unitary if and only if V is of the form

$$V=R_z(\alpha)R_y(\theta)R_z(\alpha)\sigma_x=\begin{pmatrix}\sin\theta/2 & e^{i\alpha}\cos\theta/2\\ e^{-i\alpha}\cos\theta/2 & -\sin\theta/2\end{pmatrix},$$

where α and θ are real valued.

Proof. If an additional $\wedge_1(\sigma_x)$ is appended to the end of the network in Lemma 5.4, then the network is equivalent to that above [since $\wedge_1(\sigma_x)$ is an involution] and also simulates a $\wedge_1(W\sigma_x)$ gate [since $\wedge_1(W)$ composed with $\wedge_1(\sigma_x)$ is $\wedge_1(W\sigma_x)$]. ■

Examples of matrices of the form of Lemma 5.5 are the Pauli matrices

$$\sigma_y=\begin{pmatrix}0 & -i\\ i & 0\end{pmatrix}=R_z\left(\frac{\pi}{2}\right)R_y(2\pi)R_z\left(\frac{\pi}{2}\right)\sigma_x$$

and

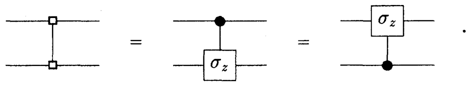
$$\sigma_z=\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}=R_z(0)R_y(\pi)R_z(0)\sigma_x$$

(as well as σ_x itself).

Lemma 5.5 permits an immediate generalization of Corollary 5.3.

Corollary 5.6. For any unitary 2×2 matrix U, a $\wedge_1(U)$ gate can be simulated by at most six basic gates: four one-bit gates $\wedge_0$ and two gates $\wedge_1(V)$, where V is of the form $V=R_z(\alpha)R_y(\theta)R_z(\alpha)\sigma_x$.

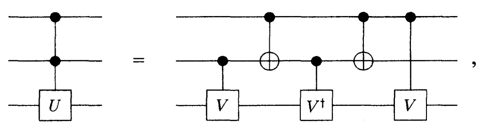
A particular feature of the $\wedge_1(\sigma_z)$ gates is that they are symmetric with respect to their input bits. In view of this, as well as for future reference, we introduce the following special notation for $\wedge_1(\sigma_z)$ gates:



VI. THREE-BIT NETWORKS

A. Simulation of general $\wedge_2(U)$ gates

Lemma 6.1. For any unitary 2×2 matrix U, a $\wedge_2(U)$ gate can be simulated by a network of the form



where V is unitary.

Proof. Let V be such that $V^2=U$. If the first bit or the second bit is 0, then the transformation applied to the third bit is either I or $VV^\dagger=I$. If the first two bits are both 1, then the transformation applied to the third is $VV=U$. ■

Some of the intuition behind the construction in Lemma 6.1 is that, using x_1 and x_2 to denote the first two input bits, the operation V is first applied to the third bit iff $x_2=1$, then $V^\dagger$ is applied if $x_1\oplus x_2=1$, then V is applied iff $x_1=1$ (iff denotes if and only if). Since

$$x_1+x_2-(x_1\oplus x_2)=2\times(x_1\wedge x_2)$$

(where "+," "−," and "×" are the ordinary arithmetic operations), the above sequence of operations is equivalent to performing V^2 on the third bit iff $x_1\wedge x_2=1$, which is the $\wedge_2(V^2)$ gate. [This approach generalizes to produce a simulation of $\wedge_m(V^{2^{m-1}})$, for $m>2$, which is considered in Sec. VII.]

We can now combine Lemma 6.1 with Corollary 5.3 to obtain a simulation of $\wedge_2(U)$ using only basic gates [$\wedge_1(\sigma_x)$ and $\wedge_0$]. The number of these gates is reduced when it is recognized that a number of the one-bit gates can be *merged* and eliminated. In particular, the $\wedge_0(C)$ from the end of the simulation of the first $\wedge_1(V)$ gate and the $\wedge_0(C^\dagger)$ from the $\wedge_1(V^\dagger)$ gate combine to form the identity and are eliminated entirely. This same sort of merging occurs to eliminate a $\wedge_0(A)$ gate and a $\wedge_0(A^\dagger)$ gate. We arrive at the following count [44].

Corollary 6.2. For any unitary 2×2 matrix U, a $\wedge_2(U)$ gate can be simulated by at most fourteen basic gates: eight one-bit gates $\wedge_0$ and six XOR gates $\wedge_1(\sigma_x)$.

A noteworthy case is when $U=\sigma_x$, where we obtain a simulation of the three-bit Toffoli gate $\wedge_2(\sigma_x)$, which is the primitive gate for classical reversible logic [4]. Later we will use the fact that because $\wedge_2(\sigma_x)$ is its own inverse, either the simulation of Lemma 6.1 or the time-reversed simulation (in which the order of the gates is reversed and each unitary operator is replaced by its Hermitian conjugate) may be used.

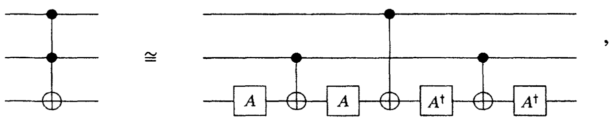
B. Three-bit gates congruent to $\wedge_2(U)$

We now show that more efficient simulations of three-bit gates are possible if phase shifts of the quantum states other than zero are permitted. If we define the matrix W as

$$W=\begin{pmatrix}0&1\\-1&0\end{pmatrix}=\Phi\left(\frac{\pi}{2}\right)\sigma_y,$$

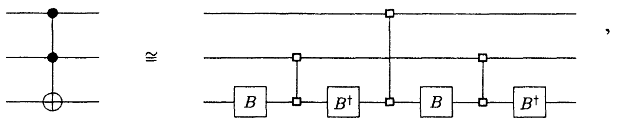
then the gates $\wedge_2(W)$ and $\wedge_2(\sigma_x)$ can be regarded as being "congruent modulo phase shifts" because the latter gate differs only in that it maps $|111\rangle$ to $-|110\rangle$ (instead of $|110\rangle$). This is perfectly acceptable if the gate is part of an operation that merely mimics classical reversible computation or if the gate is paired with another similar one to cancel out the extra phase, as is sometimes the case in reversible gate arrangements (see Corollary 7.4); however, this phase difference is dangerous in general if nonclassical unitary operations appear in the computation. Gates congruent to $\wedge_2(\sigma_x)$ modulo phase shifts have been previously investigated in [45].

The following is a more efficient simulation of a gate congruent to $\wedge_2(\sigma_x)$ modulo phase shifts:



where $A=R_y(\pi/4)$. In the above, the "≅" indicates that the networks are not identical, but differ at most in the phases of their amplitudes, which are all ±1 (the phase of the $|100\rangle$ state is reversed in this case).

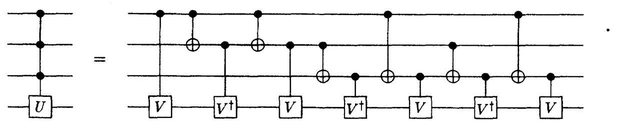
An alternative simulation of a gate congruent to $\wedge_2(\sigma_x)$ modulo phase shifts (whose phase shifts are identical to the previous one) is given by



where $B=R_y(3\pi/4)$.

VII. n-BIT NETWORKS

The technique for simulating $\wedge_2(U)$ gates in Lemma 6.1 generalizes to $\wedge_m(U)$ gates for $m>2$. For example, to simulate a $\wedge_3(U)$ gate for any unitary U, set V so that $V^4=U$ and then construct a network as follows:



The intuition behind this construction is similar to that behind the construction of Lemma 6.1. If the first three input bits are x_1, x_2, and x_3, then the sequence of operations performed on the fourth bit is

$$
\begin{aligned}
&V \quad \text{iff } x_1=1 \quad (100)\,,\\
&V^\dagger \quad \text{iff } x_1\oplus x_2=1 \quad (110)\,,\\
&V \quad \text{iff } x_2=1 \quad (010)\,,\\
&V^\dagger \quad \text{iff } x_2\oplus x_3=1 \quad (011)\,,\\
&V \quad \text{iff } x_1\oplus x_2\oplus x_3=1 \quad (111)\,,\\
&V^\dagger \quad \text{iff } x_1\oplus x_3=1 \quad (101)\,,\\
&V \quad \text{iff } x_3=1 \quad (001)\,.
\end{aligned}
$$

The strings on the right encode the condition for the operation V or $V^\dagger$ at each step: the 1's indicate which input bits are involved in the condition. For an efficient implementation of $\wedge_3(U)$, these strings form a gray code sequence. Note also that the parity of each bit string determines whether to apply V or $V^\dagger$. By comparing this sequence of operations with the terms in the equation

$$x_1+x_2+x_3-(x_1\oplus x_2)-(x_1\oplus x_3)-(x_2\oplus x_3)+(x_1\oplus x_2\oplus x_3)$$
$$=4\times(x_1\wedge x_2\wedge x_3)\,,$$

it can be verified that the above sequence of operations is equivalent to performing V^4 on the fourth bit if and only if $x_1\wedge x_2\wedge x_3=1$, which is the $\wedge_3(V^4)$ gate.

The foregoing can be generalized to simulate $\wedge_m(U)$ for larger values of m.

Lemma 7.1. For any $n\geqslant3$ and any unitary 2×2 matrix U, a $\wedge_{n-1}(U)$ gate can be simulated by an n-bit network consisting of $2^{n-1}-1$ $\wedge_1(V)$ and $\wedge_1(V^\dagger)$ gates and $2^{n-1}-2$ $\wedge_1(\sigma_x)$ gates, where V is unitary.

We omit the proof of Lemma 7.1, but point out that it is a generalization of the $n=4$ case above and based on setting V so that $V^{2^{n-2}}=U$ and "implementing" the identity

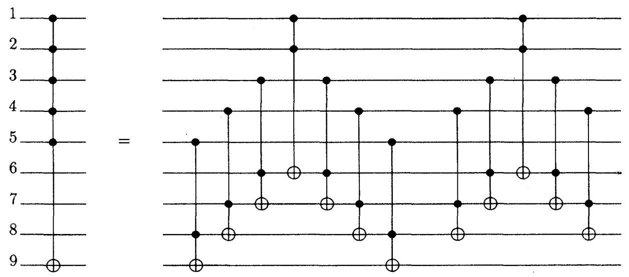
$$\sum_{k_1} x_{k_1} - \sum_{k_1<k_2} (x_{k_1}\oplus x_{k_2}) + \sum_{k_1<k_2<k_3} (x_{k_1}\oplus x_{k_2}\oplus x_{k_3}) - \cdots$$
$$+(-1)^{m-1}(x_1\oplus x_2\oplus\cdots\oplus x_m)$$
$$=2^{m-1}\times(x_1\wedge x_2\wedge\cdots\wedge x_m)$$

with a gray code sequence of operations.

For some specific small values of n (for $n=3-8$), this is the most efficient technique that we are aware of for simulating arbitrary $\wedge_{n-1}(U)$ gates as well as $\wedge_{n-1}(\sigma_x)$ gates; taking account of merges (see Corollary 6.2), the simulation requires $2\times 2^{n-1}-2\wedge_1(\sigma_x)$'s and $2\times 2^{n-1}\wedge_0$'s. However, since this number is $\Theta(2^n)$, the simulation is very inefficient for large values of n. For the remainder of this section, we focus on the asymptotic growth rate of the simulations with respect to n and show that this can be quadratic in the general case and linear in many cases of interest.

A. Linear simulation of $\wedge_{n-2}(\sigma_x)$ gates on n-bit networks

Lemma 7.2. If $n\geq 5$ and $m\in\{3,\ldots,[n/2]\}$, then a $\wedge_m(\sigma_x)$ gate can be simulated by a network consisting of $4(m-2)\wedge_2(\sigma_x)$ gates that is of the form

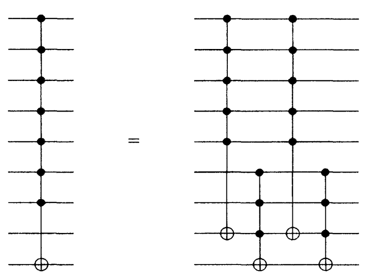


(illustrated for $n=9$ and $m=5$).

Proof. Consider the group of the first seven gates in the above network. The sixth bit (from the top) is negated if and only if the first two bits are 1, the seventh bit is negated if and only if the first three bits are 1, the eighth bit is negated if and only if the first four bits are 1, and the ninth bit is negated if and only if the first five bits are 1. Thus the last bit is correctly set, but the three preceding bits are altered. The last five gates in the network reset the values of these three preceding bits. ■

Note that in this construction and in the ones following, although many of the bits not involved in the gate are operated upon, the gate operation is performed correctly independently of the initial state of the bits (i.e., they do not have to be "cleared" to 0 first) and they are reset to their initial values after the operations of the gate (as in the computations that occur in [41] and [40]). This fact makes constructions like the following possible.

Lemma 7.3. For any $n\geq 5$ and $m\in\{2,\ldots,n-3\}$, a $\wedge_{n-2}(\sigma_x)$ gate can be simulated by a network consisting of two $\wedge_m(\sigma_x)$ gates and two $\wedge_{n-m-1}(\sigma_x)$gates, which is of the form



(illustrated for $n=9$ and $m=5$).

Proof. By inspection. ■

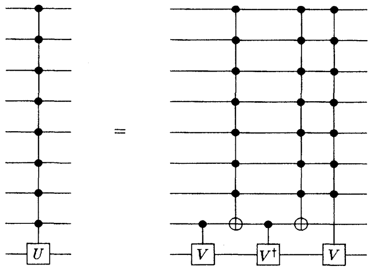
Corollary 7.4. On an n-bit network (where $n\geq 7$), a $\wedge_{n-2}(\sigma_x)$ gate can be simulated by $8(n-5)\wedge_2(\sigma_x)$ gates (three-bit Toffoli gates), as well as by $48n-212$ basic operations.

Proof. First apply Lemma 7.2 with $m_1=[n/2]$ and $m_2=n-m_1-1$ to simulate $\wedge_{m_1}(\sigma_x)$ and $\wedge_{m_2}(\sigma_x)$ gates. Then combine these by Lemma 7.3 to simulate the $\wedge_{n-2}(\sigma_x)$ gate. Then each $\wedge_2(\sigma_x)$ gate in the above simulation may be simulated by a set of basic operations (as in Corollary 6.2). We find that almost all of these Toffoli gates need only to be simulated modulo phase factors as in Sec. VI B; in particular, only four of the Toffoli gates, the ones that involve the last bit in the diagram above, need to be simulated exactly according to the construction of Corollary 6.2. Thus these four gates are simulated by fourteen basic operations, while the other $8n-36$ Toffoli gates are simulated in just seven basic operations. A careful accounting of the mergers of $\wedge_0$ and $\wedge_1$ [44] gates that are then possible leads to the total count of basic operations given above. ■

The above constructions, though asymptotically efficient, require at least one "extra" bit, in that an n-bit network is required to simulate the $(n-1)$-bit gate $\wedge_{n-2}(\sigma_x)$. In the next subsection, we shall show how to construct $\wedge_{n-1}(U)$ for an arbitrary unitary U using a quadratic number of basic operations on an n-bit network, which includes the n-bit Toffoli gate $\wedge_{n-1}(\sigma_x)$ as a special case.

B. Quadratic simulation of general $\wedge_{n-1}(U)$ gates on n-bit networks

Lemma 7.5. For any unitary 2×2 matrix U, a $\wedge_{n-1}(U)$ gate can be simulated by a network of the form



(illustrated for $n=9$), where V is unitary.

Proof. The proof is very similar to that of Lemma 6.1, setting V so that $V^2=U$. ■

Corollary 7.6. For any unitary U, a $\wedge_{n-1}(U)$ gate can be simulated in terms of $\Theta(n^2)$ basic operations.

Proof. This is a recursive application of Lemma 7.5. Let C_{n-1} denote the cost of simulating a $\wedge_{n-1}(U)$ gate (for an arbitrary U). Consider the simulation in Lemma 7.5. The cost of simulating the $\wedge_1(V)$ and $\wedge_1(V^\dagger)$ gates is $\Theta(1)$ (by Corollary 5.3). The cost of simulating the two $\wedge_{n-2}(\sigma_x)$ gates is $\Theta(n)$ (by Corollary 7.4). The cost of simulating the $\wedge_{n-2}(V)$ gate (by a recursive application of Lemma 7.5) is C_{n-2}. Therefore, C_{n-1} satisfies a recurrence of the form

$$C_{n-1}=C_{n-2}+\Theta(n) ,$$

which implies that $C_{n-1}\in\Theta(n^2)$. ■

In fact, we find that using the gate-counting mentioned in Corollary 7.4, the number of basic operations is $48n^2+\Theta(n)$. Although Corollary 7.6 is significant in that it permits any $\wedge_{n-1}(U)$ to be simulated with "polynomial complexity," the question remains as to whether a subquadratic simulation is possible. The following is an $\Omega(n)$ lower bound on this complexity.

Lemma 7.7. Any simulation of a nonscalar $\wedge_{n-1}(U)$ gate [i.e., where $U\neq\Phi(\delta)I$] requires at least $n-1$ basic operations.

Proof. Consider any n-bit network with arbitrarily many one-bit gates and fewer than $n-1$ $\wedge_1(\sigma_x)$ gates. Call two bits *adjacent* if there is a $\wedge_1(\sigma_x)$ gate between them and *connected* if there is a sequence of consecutively adjacent bits between them. Since there are fewer than $n-1$ $\wedge_1(\sigma_x)$ gates, it must be possible to partition the bits into two nonempty sets $\mathcal{A}$ and $\mathcal{B}$ such that no bit in $\mathcal{A}$ is connected to any bit in $\mathcal{B}$. This implies that the unitary transformation associated with the network is of the form $A\otimes B$, where A is $2^{|\mathcal{A}|}$ dimensional and B is $2^{|\mathcal{B}|}$ dimensional. Since the transformation $\wedge_{n-1}(U)$ is not of this form, the network cannot compute $\wedge_{n-1}(U)$. ■

It is conceivable that a linear size simulation of $\wedge_{n-1}(U)$ gates is possible. Although we cannot show this presently, in the remaining subsections we show that something "similar" (in a number of different senses) to a linear size simulation of $\wedge_{n-1}(U)$ gates is possible.

C. Linear approximate simulation of general $\wedge_{n-1}(U)$ gates on n-bit networks

Definition. We say that one network *approximates* another one *within* ε if the distance (induced by the Euclidean vector norm) between the unitary transformations associated with the two networks is at most ε.

This notion of approximation in the context of reducing the complexity of quantum computations was introduced by Coppersmith [33] and is useful for the following reason. Suppose that two networks that are approximately the same (in the above sense) are executed with identical inputs and their outputs are observed. Then the probability distributions of the two outcomes will be approximately the same in the sense that, for any event, its probability will differ by at most 2ε between the two networks.

Lemma 7.8. For any unitary 2×2 matrix U and $\varepsilon>0$, a $\wedge_{n-1}(U)$ gate can be approximated within ε by $\Theta(n\log_2(1/\varepsilon))$ basic operations.

Proof. The idea is to apply Lemma 7.5 recursively as in Corollary 7.6, but to observe that, with suitable choices for V, the recurrence can be terminated after $\Theta(\log_2(1/\varepsilon))$ levels.

Since U is unitary, there exist unitary matrices P and D such that $U=P^\dagger DP$ and

$$D=\begin{pmatrix} e^{id_1} & 0\\ 0 & e^{id_2}\end{pmatrix},$$

where d_1 and d_2 are real. e^{id_1} and e^{id_2} are the eigenvalues of U. If V_k is the matrix used in the kth recursive application of Lemma 7.3 ($k\in\{0,1,2,\dots\}$), then it is sufficient that $V_{k+1}^2=V_k$ for each $k\in\{0,1,2,\dots\}$. Thus it suffices to set $V_k=P^\dagger D_kP$, where

$$D_k=\begin{pmatrix} e^{id_1/2^k} & 0\\ 0 & e^{id_2/2^k}\end{pmatrix}$$

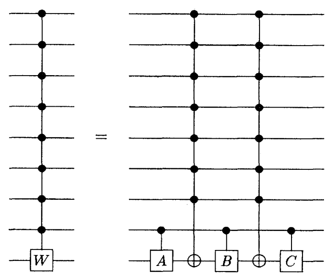
for each $k\in\{0,1,2,\dots\}$. Note that then

$$||V_k-I||_2=||P^\dagger D_kP-I||_2=||P^\dagger(D_k-I)P||_2\leq||P^\dagger||_2||D_k-I||_2||P||_2=||D_k-I||_2\leq\pi/2^k .$$

Therefore, if the recursion is terminated after $k=[\log_2(\pi/\epsilon)]$ steps, then the discrepancy between what the resulting network computes and $\wedge_{n-1}(U)$ is an $(n-k)$-bit transformation of the form $\wedge_{n-k-1}(V_k)$. Since $||\wedge_{n-k-1}(V_k)-\wedge_{n-k-1}(I)||_2=||V_k-I||_2\leq\pi/2^{[\log_2(\pi/\varepsilon)]}\leq\varepsilon$, the network approximates $\wedge_{n-1}(U)$ within ε. ■

D. Linear simulation in special cases

Lemma 7.9. For any SU(2) matrix W, a $\wedge_{n-1}(W)$ gate can be simulated by a network of the form



,

where A, B, and $C\in\mathrm{SU}(2)$.

Proof. The proof is very similar to that of Lemma 5.1, referring to Lemma 4.3. ■

Combining Lemma 7.9 with Corollary 7.4, we obtain the following.

Corollary 7.10. For any $W\in\mathrm{SU}(2)$, a $\wedge_{n-2}(W)$ gate can be simulated by $\Theta(n)$ basic operations.

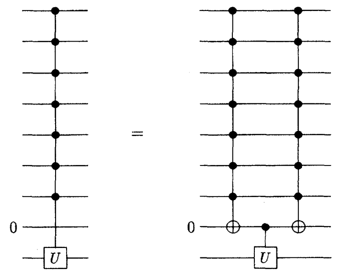
As in Sec. V, a noteworthy example is when

$$W=\begin{pmatrix} 0 & 1\\ -1 & 0\end{pmatrix}=\Phi\left(\frac{\pi}{2}\right)\sigma_y .$$

In this case, we obtain a linear simulation of a transformation congruent modulo phase shifts to the n-bit Toffoli gate $\wedge_{n-1}(\sigma_x)$.

E. Linear simulation of general $\wedge_{n-2}(U)$ gates on n-bit networks with one bit fixed

Lemma 7.11. For any unitary U, a $\wedge_{n-2}(U)$ gate can be simulated by an n-bit network of the form



(illustrated for $n=9$), where the initial value of one bit (the second to last) is fixed at 0 (and it incurs no net change).

Proof. By inspection. ■

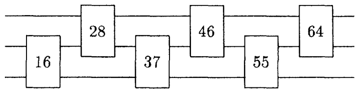
Combining Lemma 7.11 with Corollary 7.4, we obtain the following.

Corollary 7.12. For any unitary U, a $\wedge_{n-2}(U)$ gate can be simulated by $\Theta(n)$ basic operations in the n-bit network, where the initial value of one bit is fixed and incurs no net change. Note that the "extra" bit above may be reused in the course of several simulations of $\wedge_m(U)$ gates.

VIII. EFFICIENT GENERAL GATE CONSTRUCTIONS

In this final discussion we will change the ground rules slightly by considering the "basic operation" to be *any* two-bit operation. This may or may not be a physically reasonable choice in various particular implementations of quantum computing, but for the moment this should be considered as just a mathematical convenience that will permit us to address somewhat more general questions than the ones considered above. When the arbitrary two-bit gate is taken as the basic operation, then as we have seen, five operations suffice to produce the Toffoli gate (recall Lemma 6.1), three produce the Toffoli gate modulo phases (we permit a merging of the operations in the construction of Sec. VI B), and thirteen can be used to produce the four-bit Toffoli gate (see Lemma 7.1). In no case do we have a proof that this is the most economical method for producing each of these functions; however, for most of these examples we have compelling evidence from numerical study that these are in fact minimal [45].

In the course of doing these numerical investigations we discovered a number of interesting additional facts about two-bit gate constructions. It is natural to ask, how many two-bit gates are required to perform *any arbitrary* three-bit unitary operation, if the two-bit gates were permitted to implement any member of U(4)? The answer is six, as in the gate arrangement shown here:



.

We find an interesting regularity in how the U(8) operation is built up by this sequence of gates, which is summarized by the "dimensionalities" shown in the diagram. The first U(4) operation has $4^2=16$ free angle parameters; this is the dimensionality of the space accessible with a single gate, as indicated. With the second gate, this dimensionality increases only by 12, to 28. It does not double to 32, for two reasons. First, there is a single global phase shared by the two gates. Second, there is a set of operations acting only on the bit shared by the two gates, which accounts for the additional reduction of 3. Formally, this is summarized by noting that 12 is the dimension of the *coset space* SU(4)/SU(2). The action of the third gate increases the dimensionality by another $9=16-1-3-3$. The dimension of the coset space SU(4)/SU(2)$\times$SU(2) is 9. The further subtraction by 3 results from the duplication of one-bit operations on both bits of the added gate. At this point the dimensionality increases by 9 for each succeeding gate, until the dimensionality reaches exactly 64, the dimension of U(8), at the sixth gate. In preliminary tests on four-bit operations, we found that the same rules for the increase of dimensionality applied. This permits us to make a conjecture, based just on dimension counting, of a lower bound $\frac{1}{9}4^n-\frac{1}{3}n-\frac{1}{9}=\Omega(4^n)$ on the number of two-bit gates required to produce an arbitrary n-bit unitary transformation. It is clear that "almost all" unitary transformations will be computationally uninteresting since they will require exponentially many operations to implement.

Finally, we mention that by combining the quantum gate constructions introduced here with the decomposition formulas for unitary matrices as used by Reck *et al.* [15], an explicit, exact simulation of *any* unitary operator on n bits can be constructed using a finite number $[\Theta(n^3 4^n)]$ of two-bit gates and using no work bits. In outline, the procedure is as follows. Reck *et al.* [15] note that a formula exists for the decomposition of any unitary matrix into matrices only involving a U(2) operation acting in the space of pairs of *states* (not *bits*):

$$U=\left(\sum_{x_1,x_2\in\{0,1\}^m,x_1>x_2} T(x_1,x_2)\right)D\ .$$

$T(x_1,x_2)$ performs a U(2) rotation involving the two basis states x_1 and x_2 and leaves all other states unchanged; D is a diagonal matrix involving only phase factors and thus can also be thought of as a product of 2^{n-1} matrices that perform rotations in two-dimensional subspaces. Using the methods introduced above, each $T(x_1,x_2)$ can be simulated in polynomial time as follows. Write a gray code connecting x_1 and x_2; for example, if $n=8$, $x_1=00111010$ and $x_2=00100111$, we have

1	00111010	x_1
2	00111011	
3	00111111	
4	00110111	
5	00100111	x_2

Operations involving adjacent steps in this gray code require a simple modification of the $\wedge_{n-1}$ gates introduced earlier. The $(n-1)$ control bits that remain unchanged are not all 1 as in our earlier constructions, but they can be made so temporarily by the appropriate use of "NOT" gates $[\wedge_0(\sigma_x)]$ before and after the application of the $\wedge_{n-1}$ operation. Now, the desired $T(x_1,x_2)$ operation is constructed as follows. First, permute states down through the gray code, performing the permutations (1,2), (2.3), (3,4), . . . ,$(m-2,m-1)$. These numbers refer to the gray code elements as in the table above, where m, the number of elements in the gray code, is 5 in the example. Each of these permutations is accomplished by a modified $\wedge_{n-1}(\sigma_x)$. Second, the desired U(2) rotation is performed by applying a modified $\wedge_{n-1}(U)$ involving the states $(m-1)$ and (m). Third, the permutations are undone in reverse order: $(m-2,m-1)$, $(m-3,m-2)$, . . . ,(2,3), (1,2).

The number of basic operations to perform all these steps may be easily estimated. Each $T(x_1,x_2)$ involves $2m-3$ (modified) $\wedge_{n-1}$ gates, each of which can be done in $\Theta(n^2)$ operations. Since m, the number of elements in the gray code sequence, cannot exceed $n+1$, the number of operations to simulate $T(x_1,x_2)$ is $\Theta(n^3)$. There are $\Theta(4^n)$ T's in the product above, so the total number of basic operations to simulate any U(2^n) matrix exactly is $\Theta(n^3 4^n)$. (The number of steps to simulate the D matrix is smaller and does not affect the count.) So we see that this strict upper bound differs only by a polynomial factor (which likely can be made better than n^3) from the expected lower bound quoted earlier, so this Reck procedure is relatively "efficient" (if something that scales exponentially may be termed so). A serious problem with this procedure is that it is extremely unlikely, as far as we can tell, to provide a polynomial-time simulation of those special U(2^n) that permit it, which of course are exactly the ones that are of most interest in quantum computation. It still remains to find a truly efficient and useful design methodology for quantum gate construction.

Note added in proof: E. Knill (unpublished) has recently improved the upper bound of Sec. VIII from $\Theta(n^3 4^n)$ to $\Theta(n 4^n)$, and confirmed the lower bound that we give.

We are very grateful to H.-F. Chau, D. Coppersmith, D. Deutsch, A. Ekert, and T. Toffoli for helpful discussions. We are also most happy to thank the Institute for Scientific Interchange in Torino and its director Professor Mario Rasetti for making possible the workshop on quantum computation at which much of this work was performed. A.B. thanks the financial support of the Berrow's fund in Lincoln College (Oxford).

[1] P. A. M. Dirac, *The Principles of Quantum Mechanics* (Oxford University Press, New York, 1958), Chap. 5; for a modern treatment, see A Peres, *Quantum Theory: Concepts and Methods* (Kluwer, Dordrecht, 1993), Chap. 8.6.
[2] R. Landauer, IBM J. Res. Dev. **5**, 183 (1961); C. H. Bennett, *ibid.* **17**, 525 (1973); Yves Lecerf, C. R. Acad. Sci. **257**, 2597 (1963); C. H. Bennett, SIAM J. Comput. **18**, 766 (1989); for a review, see C. H. Bennett and R. Landauer, Sci. Am. **253** (1), 48 (1985).
[3] E. Fredkin and T. Toffoli, Int. J. Theor. Phys. **21**, 219 (1982).
[4] T. Toffoli, in *Automata, Languages and Programming*, edited by J. W. de Bakker and J. van Leeuwen (Springer, New York, 1980), p. 632; MIT Technical Memo No. MIT/LCS/TM-151, 1980 (unpublished).
[5] P. Benioff, J. Stat. Phys. **29**, 515 (1982).
[6] A. Peres, Phys. Rev. A **32**, 3266 (1985).
[7] R. P. Feynman, Opt. News **11** (2), 11 (1985).
[8] N. Margolus, in *Complexity, Entropy, and the Physics of Information*, edited by W. H. Zurek (Addison-Wesley, Reading, MA, 1990), Vol. VIII, p. 273.
[9] Tycho Sleator and Harald Weinfurter, Phys. Rev. Lett. **74**, 4087 (1995).
[10] A. Barenco, D. Deutsch, A. Ekert, and R. Jozsa, Phys. Rev. Lett. **74**, 4083 (1995).
[11] J. I. Cirac and P. Zoller, Phys. Rev. Lett. **74**, 4091 (1995).
[12] I. Chuang and Y. Yamamoto (unpublished).
[13] Y. Yamamoto, M. Kitegawa, and K. Igeta, in *Proceedings of the Third Asia-Pacific Physics Conference*, edited by Y. W. Chan *et al.* (World Scientific, Singapore, 1988), p. 779; G. J. Milburn, Phys. Rev. Lett. **62**, 2124 (1989).
[14] S. Lloyd, Science **261**, 1569 (1993); **263**, 695 (1994).
[15] M. Reck, A. Zeilinger, H. J. Bernstein, and P. Bertani, Phys. Rev. Lett. **73**, 58 (1994); see also Francis D. Murnaghan, *The Unitary and Rotation Groups* (Spartan, Washington, D.C., 1962), Chap. 2.
[16] D. Deutsch, Proc. R. Soc. London Ser. A **400**, 97 (1985).
[17] D. Deutsch and R. Jozsa, Proc. R. Soc. London Ser. A **439**, 553 (1992).
[18] A. Berthiaume and G. Brassard, in *Proceedings of the Workshop on Physics and Computation, PhysComp '92* (IEEE Computer Society, Los Alamitos, 1992), pp. 195–199.
[19] D. Simon, in *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science* (IEEE Computer Society, Los Alamitos, 1994), p. 116.
[20] E. Bernstein and U. Vazirani, in *Proceedings of the 25th Annual ACM Symposium on Theory of Computing* (ACM, New York, 1993), pp. 11–20.
[21] P. W. Shor, in *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science* (Ref. [19]), p. 124.
[22] For a review see A. Ekert and R. Jozsa (unpublished); see also J. Brown, New Sci. **133**, 21 (1994).
[23] For an interesting recent perspective see G. Brassard, SIGACT News **25**, 15 (1994).
[24] D. Deutsch, Proc. R. Soc. London Ser. A **425**, 73 (1989).
[25] B. Schumacher, Phys. Rev. A **51**, 2738 (1995).
[26] R. Jozsa and B. Schumacher, J. Mod. Opt. **41**, 2343 (1994).
[27] A. C.-C. Yao, in *Proceedings of the 34th Annual Symposium on the Foundations of Computer Science* (IEEE Computer Society, Los Alamitos, 1993), p. 352.
[28] D. P. DiVincenzo, Phys. Rev. A **50**, 1015 (1995).
[29] A. Barenco, Proc. R. Soc. London Ser. A **449**, 678 (1995).
[30] S. Lloyd, Phys. Rev. Lett. **75**, 346 (1995).
[31] D. Deutsch, A. Barenco, and A. Ekert, Proc. R. Soc. London Ser. A. **449**, 669 (1995).
[32] H. F. Chau and F. Wilczek, Phys. Rev. Lett. **50**, 748 (1995).
[33] D. Coppersmith, IBM Research Report No. RC19642, 1994 (unpublished); R. Cleve (unpublished).
[34] A. Berthiaume, D. Deutsch, and R. Jozsa, *Proceedings of the Workshop on Physics and Computation, PhysComp '94* (IEEE, Los Alamitos, 1994), p. 60.

[35] W. G. Unruh, Phys. Rev. A **51**, 992 (1995).

[36] I. L. Chuang, R. Laflamme, P. Shor, and W. H. Zurek (unpublished).

[37] R. Landauer, Trans. R. Soc. London (to be published).

[38] The only nontrivial one-bit classical invertible operation is $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$.

[39] D. Coppersmith and E. Grossman, SIAM J. Appl. Math. **29** (4), 624 (1975).

[40] Richard Cleve, Ph.D. thesis, University of Toronto, 1989 (unpublished), pp. 2–56.

[41] Michael Ben-Or and Richard Cleve, SIAM J. Comput. **21**, 54 (1992).

[42] Corresponding to the definition in Sec. II, the top two wires contain the input bits x_1 and x_2 and the third wire contains the input y.

[43] The angle-halving of this definition conforms to the usual relation between operations in SO(3) and SU(2). See J. Mathews and R. L. Walker, *Mathematical Methods of Physics*, 2nd ed. (Benjamin, Menlo Park, CA, 1970), p. 464 for the use of SO(3) language (rigid body rotations) to describe SU(2) operations.

[44] These counts take advantage of a different, entirely classical kind of merging. By an inspection of the truth table of the sequences of the three consecutive $\wedge_1(\sigma_x)$ gates that appear in these simulations, we find that they can always be replaced by just two XOR gates. In one of these cases, in order to make the three $\wedge_1(\sigma_x)$ gates consecutive, an interchange of the order of an XOR gate with a one-bit gate of the "E" type from Lemma 5.2 must be done. (The two gates commute.)

[45] D. P. DiVincenzo and J. Smolin, in *Proceedings of the Workshop on Physics and Computation, PhysComp '94* (Ref. [34]), p. 14.