REVERSIBLE CIRCUIT REALIZATIONS OF BOOLEAN FUNCTIONS

Alex Brodsky
Department of Computer Science
University of Toronto, Canada
abrodsky@cs.utoronto.ca

Abstract Reversible circuits are a concrete model of reversible computation with applications in areas such as quantum computing and analysis of cryptographic block cyphers. In 1980, Toffoli showed how to realize a Boolean function by a reversible circuit, however the resulting complexity of such circuits has remained an open problem. We investigate the reversible circuit complexity of families of Boolean functions and derive conditions that characterize whether a polynomial realization is possible.

First, we derive sufficient conditions on families of Boolean functions that guarantee a polynomial-size reversible circuit realization. Namely, we show that if a Boolean function can be embedded into an even parity permutation that has a polynomial-size cycle representation, then the Boolean function can be realized by a polynomial-size reversible circuit. Furthermore, we provide a construction for the realization. Second, we provide concrete realizations for several families of Boolean functions, such as the adder, incrementor, and threshold functions, which do not necessarily satisfy the preceding condition, but still have polynomial-size realizations; this is important because such realizations will necessarily form the building blocks of quantum computers.

Keywords: Reversible computation, circuit complexity, Boolean functions

1. Introduction

Reversible circuits, introduced by Landauer [Lan61] and formalized by Toffoli and Fredkin [Tof80, FT82], are a concrete model of reversible computation that have come to prominence in the last few years; their applications range from quantum computing [BBD+95], where reversibility is a prerequisite, to analysis of cryptographic block cyphers [Cle90, EG83], which use primitives that are nearly identical to those comprising reversible circuits. Reversible computation is based on a notion of equivalence between information and entropy that was formalized by Shannon and Weaver [Sha48, SW49], but dates back to Maxwell's Demon [Max71] and the work of Szilard [Szi29]. Namely, the operations comprising a reversible computation may not discard any information during the course of the computation. In this paper we in-

vestigate the reversible circuit complexity of families of Boolean functions and derive conditions that characterize whether a polynomial realization is possible.

Reversible circuits on n lines (wires) realize permutations on the Boolean cube of dimension n. In 1980, Toffoli [Tof80] showed how to embed Boolean functions into permutations and thus, be realizable by a reversible circuit. Namely, an n-adic Boolean function can be embedded into a permutation on a Boolean cube of dimension $n+1$. However, just as in the case of classical circuit complexity, the complexity of the corresponding reversible circuit is difficult to determine. Some results were obtained by Cleve [Cle90], who showed that polynomial length compositions of D.E.S.-like cipher functions—function generators, of fan-in 2—can compute $\mathbf{NC}^1$. The construction is reminiscent of Barrington's [Bar86] proof that width-5 permutation branching programs compute $\mathbf{NC}^1$. The point of Cleve's investigation was to determine if such ciphers could be used as pseudorandom generators.

Alternatively, since most Boolean functions are irreversible, the reversible circuit complexity of a Boolean function is directly related to the complexity of simulating irreversible computation reversibly. Bennett [Ben73] first described two simulation techniques, within the context of Turing machines, that used additional space to record a check-point based history of the simulation. The first simulation used $O(T)$ time and $O(S+T)$ space to reversibly simulate a computation that takes T time and S space; the second simulation used $O(T^2)$ time and $O(S\log T) \subseteq O(S^2)$ space. The latter simulation was later refined to use $O(T^{1+\varepsilon})$ time and $O(S\log T)$ space [Ben89, LS90]. Although Bennett's constructions are of the same spirit as the circuit constructions of Toffoli [Tof80], it is the space parsimonious reversible simulation of a Turing machine by Lange et al. [LMT00] that mostly closely resembles the n-line reversible circuit model.

We show that any even parity permutation on an n-dimensional Boolean cube whose cycle representation is of size s can be realized by a reversible circuit of size $O(sn)$. The key corollary is that any Boolean function that can be embedded into a permutation with a polynomial size cycle representation, can be realized by polynomial size reversible circuit. Furthermore, the proof is completely constructive, yielding a simple methodology for designing reversible circuits.

In many cases this bound is not tight because there are many families of functions, such as the incrementor, whose corresponding permutations have exponentially large cycle representations, but polynomial-size circuit realizations. We exhibit several families of functions with such characteristics and derive realizations for them. Particularly, we focus on functions that are commonly implemented in hardware and will necessarily need to be implemented as part of a quantum computer, i.e., reversibly. We consider several families of functions, including incrementors, adders, consensus, and threshold functions.

In Section 2 we formally define the reversible circuit model and describe how Boolean functions are embedded within permutations on the Boolean cube. In Section 3 we prove our main result and in Section 4 we provide concrete constructions for several families of Boolean functions. Section 5 summarizes some of the techniques for constructing reversible circuits and finally, section 6 places our results in the greater context and provides some future directions.

2. Background

Reversible circuits comprise a number of wires, called **lines**, and reversible **gates** that operate on the lines. The lines carry binary values, 0 or 1, which are placed on the lines' input terminals, are modified by gates operating on the lines, and are read off the lines' output terminals; by convention, the input terminals are on the left side and the output terminals are on the right (see Figure 1). Each gate operates on at most three lines. All but one of the lines pass through the gate unmodified and are called **control lines**. The remaining line, called the **toggle line** is XORed by the gate with the conjunction of the values of the control lines. Each gate realize a bijection on the Boolean cube and each gate is also it's own inverse.

Let $B_n = \{0,1\}^n$ denote the Boolean cube of dimension n, let $x \in B_n$ denote an n-bit vector, and let x_i denote the ith bit of x. A reversible circuit C, on n lines, is specified by a sequence of m gates, $C = g_1 g_2 \ldots g_m$: the gates are the NOT gate, denoted $\bigoplus_i$; the controlled-NOT gate, denoted $\bigoplus_i^j$; and the Toffoli gate, denoted $\bigoplus_i^{j \wedge k}$, where $j,k \in [n]$ specify the control lines and $i \in [n]$ specifies the toggle line. In the nomenclature of Coppersmith and Grossman [CG75], the three gates correspond to the 0-, 1-, and 2-functions, respectively. For example, the circuit in Figure 1 comprises two Toffoli gates and two NOT gates.

The output of circuit C on input x, is denoted $C(x)$, and the composition and inverse of circuits corresponds respectively to the concatenation and reversal of the circuits' gate sequences; we write CC' to denote the concatenation of two circuits and C^{-1} to denote the inverse of C. Each reversible circuit realizes a permutation on the Boolean cube B_n, corresponding to an element of the symmetric group S_{2^n}. We write $C \sim \sigma \in S_{2^n}$ if C realizes permutation σ and we write $C \sim C'$ if C and C' realize the same permutation, e.g., $C \sim (1\,2\,3)$. For conciseness, we assume that circuit $C_{(xyz)} \sim (xyz)$, $x,y,z \in B_n$ and in general that circuit $C_\sigma \sim \sigma$.

We often use a notion of a controlled circuit in our constructions. An (i)-**controlled** circuit, denoted $C^{(i)}$, performs two different permutations depending on the value of line i, leaving the value of line i unchanged. If line i has value 1, the circuit performs a fixed permutation, otherwise the circuit performs the identity permutation. Analogously, a $\overline{(i)}$-**controlled** circuit performs a fixed permutation only if the value of line (i) is 0. In general, a circuit is x-**controlled**, for some $x \in B_k$ and $k < n$, if the circuit, denoted C^x, is controlled by a fixed subset of control lines of size k. If the lines hold the value x, then C^x performs a fixed permutation, leaving the control lines unchanged, and performs the identity permutation otherwise. For example, the circuit in Figure 1 is a $\overline{(3)}$-controlled circuit,

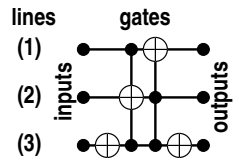

Figure 1. The circuit $C = \bigoplus_3 \bigoplus_2^{1\wedge 3} \bigoplus_1^{2\wedge 3} \bigoplus_3 \sim (1\,2\,3)$.

For conciseness, we use several schematic short forms. First, the k-line controlled Toffoli gate, $k < n-1$, which computes the conjunction of k lines and XORs the output line. A k-line controlled Toffoli (k-Toffoli) gate can be constructed using $O(k)$ Toffoli gates [BBD+95] and is illustrated in Figure 2a. Second, the controlled k-NOT, comprises k controlled-NOT gates that are all controlled by the same line. In most cases, we will be using the controlled $(n-1)$-NOT, which is illustrated in Figure 2c. Additionally, we use blocks to denote a component of a circuit: a component may either be **simple** (Figure 2e); **controlled** (Figure 2d), i.e., controlled by another line; or a k-**function** (Figure 2b), i.e., XOR a line with a Boolean function computed on k other lines. The controlled k-NOT and the k-Toffoli gate are examples of a controlled component and a k-function.

The **size** of circuit C, denoted $|C|$ is the number of gates comprising C. The **depth** of C, denoted $d(C)$ is the length of the longest path through the directed acyclic graph induced by circuit C: the lines correspond to right-oriented arcs, the gates correspond to vertices of equal indegree and outdegree, the input terminals correspond to vertices with indegree 0, and the output terminals correspond to vertices of outdegree 0. Two gates are **pairwise independent** if there is no path from one to the other in the induced graph. Since the number gates that are all pairwise independent is at most n, the depth of a circuit is at most a factor of n less than the size, i.e., $|C|/n \leq d(C) \leq |C|$. Finally, the size of the cycle representation of a permutation σ, denoted $|\sigma|$, is the number of points in the permutation that are not fixed, e.g., a transposition has size 2.

2.1 Embedding Boolean Functions

To realize a Boolean function by a reversible circuit, the function must first be embedded into a permutation, because reversible circuits can only realize permutations on the Boolean cube. In the spirit of Toffoli [Tof80], an $(n+c)$-**embedding** of an n-adic Boolean function f is a permutation σ on an $n+c$ dimensional Boolean cube such that if $y = \sigma x$, for any input x (padded with zeros), then $y_{n+c} = f(x)$. The embedding is said to be **input-preserving** if $x_i = y_i$, for all $i \in [n]$, i.e., the embedding preserves the input.

In many physical contexts, such as quantum computation, using additional lines—in addition to the n lines containing the input—is expensive. Thus, we restrict our attention to embeddings that use the minimum number of additional lines: $c \leq 2$. Unless f is linear in some variable x_i, i.e., $f(x) = g(x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n) \oplus x_i$, an

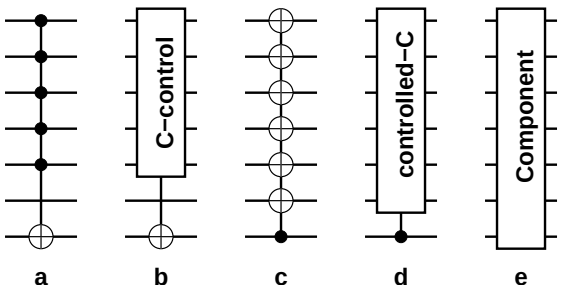

Figure 2
a) A k-Toffoli gate;
b) a k-function component;
c) a controlled n-NOT;
d) a controlled component;
e) a simple component.

input-preserving n-embedding of f does not exist; and, if f is unbalanced, there does not exist any kind of n-embedding of f [Tof80]. Thus, in most cases we consider realizations of $(n+1)$-embeddings, i.e., $c = 1$. In fact, if we require an input-preserving embedding of an unbalanced function with an odd number of satisfying assignments, only an $(n+2)$-embedding will suffice.

If f is Boolean function of the form $f : B_n \to B_k$, $1 < k \leq n$, a similar criterion can be derived. However, many such functions do not have an $(n+c)$-embedding, where c is a constant [Tof80]. For example, the multiplication function, which takes two n-bit strings and yields a $2n$ bit string, does not have a m-embedding where $m < 4n$!

On two occasions we consider functions that are of the form $f : B_n \to B_n$, e.g., the incrementor (counter) function that maps $z \to z+1 \bmod 2^n$. In this case, even if f is a bijection, a corresponding n-embedding does not exist because the corresponding permutation has odd parity. For $n > 3$, odd parity permutations cannot be realized by a reversible circuit that comprises NOT, controlled-NOT, and Toffoli gates [CG75]. However, an $(n+1)$-embedding of even parity is possible, and hence can be realized by a reversible circuit on $n+1$ lines.

The reversible circuit complexity of a Boolean function f, is the minimum over all circuit realizations of all possible embeddings of f. However, determining reversible circuit complexity of realizing a permutation is not obvious. In the next section we characterize families of permutations, and hence families of Boolean functions, that have reversible circuit complexity which is polynomial in n.

3. The Main Result

In this section we prove the following theorem:

THEOREM 1 *Any even parity permutation σ on B_n, can be realized by an n-line circuit of size $O(|\sigma|n)$.*

The proof comprises three parts: first, we prove that given a circuit that realizes a permutation that is represented by a single 3-cycle, the circuit can be modified to realize any other 3-cycle and that the modifications can be accomplished with $O(n)$ gates. Second, we show that the 3-cycle $(0\,1\,2)$ can be realized by a reversible circuit on $n > 1$ lines that is of size $O(n)$. Third, we note that any even parity permutation whose cycle representation is of size s can be factored into $O(s)$ 3-cycles. Combining these three facts yields the result. The following lemma and its corollary form the heart of the first part of the proof, which is summarized in Theorem 4.

LEMMA 2 *Let $C_{(0\,1\,2)}$ be a reversible circuit on n lines. For any $x,y \in B_n$, $x \neq y$ and $x,y \neq 0$, there exists a reversible circuit C of size $O(n)$, such that the circuit $CC_{(0\,1\,2)}C^{-1} \sim C_{(0xy)}$.*

Proof: Select two lines i and j, setting $u = x_i x_j$, $v = y_i y_j$, $u,v \in B_2$, such that $u \neq v$ and $u,v \in \{1,2,3\}$. Such a choice is possible, otherwise $x = 0$, $y = 0$, or $x = y$, none of which can happen because $(0xy)$ is a 3-cycle. Call lines i and j the control lines. The circuit C consists of three stages. Stage one comprises $|x| - |u|$ Toffoli gates plus $|y| - |v|$ Toffoli gates. The first subsequence is bracketed by a pair of NOT gates on line i (j) if x_i (x_j) is 0; the second sequence is analogously bracketed if y_i (y_j) is 0. For

each $k \neq i, j$, if $x_k = 1$ a Toffoli gate $\bigoplus_k^{i \wedge j}$, controlled by lines i and j, toggles line k. The second subsequence of Toffoli gates is analogously specified. Thus, on input x or y, all lines but lines i and j are toggled to 0.

Stage two swaps line i with line 1 and line j with line 2. This can be done using $O(1)$ gates. Finally, stage three manipulates lines 1 and 2 since these lines now hold the value of the control lines. If $u \neq 1$ and $v \neq 2$, then stage three maps u to 1 and v to 2; this also takes $O(1)$ gates. Therefore, circuit C maps input 0 to 0, input x to 1, and input y to 2, using $O(|x| + |y|) \subseteq O(n)$ gates. The circuit may permute other points in B_n, but this is of no consequence.

Since $C \sim (x1 \ldots y2 \ldots)$, composing circuit C with $C_{(012)}$ in the form of a conjugate yields

$$CC_{(012)}C^{-1} \sim (x1 \ldots y2 \ldots)(012)(x1 \ldots y2 \ldots)^{-1} = (0xy) \sim C_{(0xy)},$$

which completes the proof. ■

COROLLARY 3 *Let $C_{(0xy)}$ be a reversible circuit on n lines. There exists a reversible circuit C of size $O(n)$, such that the circuit $CC_{(0xy)}C^{-1} \sim C_{(012)}$.*

Theorem 4 follows easily from the lemma and the corollary.

THEOREM 4 (3-CYCLE HARDNESS THEOREM) *If $C_{(xyz)}$ is a reversible circuit on n lines, then for any distinct $x', y', z' \in B_n$, there exists a circuit C of size $O(n)$, such that $CC_{(xyz)}C^{-1} \sim C_{(x'y'z')}$.*

Proof: Since XORing the input with a constant bit vector can be performed by $O(n)$ NOT gates, we can transform $C_{(xyz)}$ into $C_{0\bar{y}\bar{z}}$, where $\bar{y} = y \oplus x$ and $\bar{z} = z \oplus x$. Similarly, for a circuit $C_{(0\bar{y}'\bar{z}')}$, where $\bar{y}' = y' \oplus x'$ and $\bar{z}' = z' \oplus x'$, can be transformed into $C_{(x'y'z')}$ using $O(n)$ gates. Let C_x be the circuit comprising $|x|$ NOT gates such that $C_x C_{(xyz)} C_x^{-1} \sim C_{(0\bar{y}\bar{z})}$ and correspondingly let $C_{x'}$ be such that $C_{(x'y'z')} \sim C_{x'} C_{(0\bar{y}'\bar{z}')} {C_{x'}}^{-1}$.

By Corollary 3, $C_{(0\bar{y}\bar{z})}$ can be transformed into $C_{(012)}$ and by Lemma 2, this circuit can be transformed into $C_{(0\bar{y}'\bar{z}')}$, also in $O(n)$ gates. Let C_1 and C_2 be the circuits such that $C_{(012)} \sim C_1 C_{(0\bar{y}\bar{z})} C_1^{-1}$ and $C_{(0\bar{y}'\bar{z}')} \sim C_2 C_{(012)} C_2^{-1}$. Since

$$C_{(x'y'z')} \sim C_{x'} C_2 C_1 C_x C_{(xyz)} C_x^{-1} C_1^{-1} C_2^{-1} {C_{x'}}^{-1},$$

setting $C = C_{x'} C_2 C_1 C_x$, which is of size $O(n)$, completes the proof. ■

Thus, all 3-cycles are equally hard to realize in the sense that, given a polynomial size realization of one 3-cycle, any other 3-cycle can be realized by using an additional $O(n)$ gates. The second step of the proof, Theorem 5, shows how a 3-cycle can be realized by a reversible circuit of size $O(n)$.

THEOREM 5 *For $n > 1$ there is an n-line reversible circuit $C_{(012)}$ of size $O(n)$.*

Proof: If $1 < n \leq 3$ we can construct a reversible circuit that realizes any permutation and uses a constant number of gates [CG75].

For $n > 3$, observe that permutation $(0\,1\,2)$ may be factored into $\tau_1 = (0\,1)(6\,3)$ and $\tau_2 = (0\,2)(6\,3)$. Thus, we need only demonstrate that permutations τ_1 and τ_2 can be realized in $O(n)$ gates.

First, the permutation $(0\,1)(2\,3)$ (respectively $(0\,2)(1\,3)$) may be realized by $O(n)$ gates. The circuit comprises three stages: a negation, followed by a toggling, followed by a negation; each stage requires $O(n)$ gates. Stages one and three negate the $n-2$ lines $3,\ldots,n$. The middle stage comprises an $(n-2)$-Toffoli gate, controlled by lines $3,\ldots,n$, that toggles line 1 (respectively line 2). Let $C_{(0\,1)(2\,3)} \sim (0\,1)(2\,3)$ and $C_{(0\,2)(1\,3)} \sim (0\,2)(1\,3)$ respectively.

Next, we construct a reversible circuit $C_{(0\,1)(6\,3)}$ that realizes permutation $(0\,1)(6\,3)$. The reversible circuit $C_{\sigma_1} = \bigoplus_1 \bigoplus_3^{1\wedge 2} \bigoplus_1$ realizes a permutation that transposes 2 and 6 and whose fixed-points include all points that are congruent to 0, 1, or 3 modulo 4. Since $\sigma_1(0\,1)(2\,3)\sigma_1 = (0\,1)(6\,3)$, therefore $C_{(0\,1)(6\,3)} = C_{\sigma_1} C_{(0\,1)(2\,3)} {C_{\sigma_1}}^{-1}$.

Similarly, we construct $C_{(0\,2)(6\,3)}$. The circuit $C_{\sigma_2} = \bigoplus_2 \bigoplus_3^{1\wedge 2} \bigoplus_2$ transposes points 1 and 5, with the fixed-points comprising all points that are congruent to 0, 2, and 3 modulo 4. Using conjugation, we construct circuit $C_{(0\,2)(5\,3)} = C_{\sigma_2} C_{(0\,2)(1\,3)} C_{\sigma_2}$. The circuit $C_\rho = \bigoplus_1^{2\wedge 3} \bigoplus_2^{1\wedge 3} \bigoplus_1^{2\wedge 3}$, switches the values of the lines 1 and 2, using line 3 as the control. Permutation ρ transposes 5 and 6; only points congruent to 5 or 6 modulo 8 are permuted. Since $\rho(0\,2)(5\,3)\rho^{-1} = (0\,2)(6\,3)$, therefore $C_{(0\,2)(6\,3)} = C_\rho C_{(0\,2)(5\,3)} C_\rho^{-1}$.

The required circuit is $C_{(0\,1\,2)} = C_{(0\,1)(6\,3)} C_{(0\,2)(6\,3)}$ is of size $O(n)$. ■

In conjunction with Theorem 4, we get the following two corollaries.

COROLLARY 6 *Any* 3*-cycle can be realized by a reversible circuit on* $n > 1$ *lines of size* $O(n)$.

COROLLARY 7 *A permutation on* B_n *comprising two disjoint transpositions, can be realized by a reversible circuit on n lines of size* $O(n)$.

Proof: This follows from the fact that $(a\,b)(c\,d) = (a\,b\,c)(c\,a\,d)$. ■

We note that every even parity permutation σ can be factored into $|\sigma| - 1$ transpositions and that by Corollary 7, any pair of transpositions can be realized with $O(n)$ gates. Hence, every even parity permutation σ has a reversible circuit realization of size $O(|\sigma| n)$, which is the statement of Theorem 1. Thus, any Boolean function that has an embedding with a polynomial size cycle representation can be realized by a polynomial size reversible circuit. Unfortunately, the converse is not true. There are many families of Boolean functions, such as the negation of a projection, and the incrementor, that have an exponential size cycle representation but a concise reversible circuit realization. In the next section we detail realizations for several of such families of functions.

4. Applications: Concrete Realizations

Two common families of functions that are ubiquitous in digital circuits are the incrementor family, which includes the decrementor and the adder, and the threshold family, which includes such variants as the conjunction, the disjunction, the majority,

and the consensus. We first describe how to realize an incrementor, or more precisely, a near approximation of one. The adder can then be built from a number of incrementors.

The incrementor presents an interesting challenge for several reasons. First, its cycle representation is large, comprising all 2^n points of the Boolean cube. Second, even though the incrementor is a bijection, it cannot be realized (in full) by a reversible circuit because the corresponding permutation has odd parity [CG75]. Thus, at best, the incrementor can only be approximated by a reversible circuit. Even though the cycle representation is large, various approximations of the incrementor can be realized efficiently: our realization is of size and depth $O(n^2)$.

4.1 Realization of Various Incrementors

Since a full incrementor on B_n is impossible [CG75], we begin by constructing a **half-incrementor** that performs a full increment on the subspace B_{n-1} and is represented by two disjoint cycles of the form

$$\pi = (0\,1 \ldots 2^{n-1}-1)(2^{n-1}\,2^{n-1}+1 \ldots 2^n-1).$$

By concatenating this realization with another small circuit, we construct a **nigh-incrementor**, which corresponds to the permutation

$$\pi' = (0\,1 \ldots 2^{n-1}-2),$$

i.e., performs the operation $z \to z+1 \bmod 2^n-1$ rather than $\bmod 2^n$.

The half-incrementor can be realized via a sequence of k-Toffoli gates, where $k = n-2 \ldots 0$. Observe that an incrementor modifies the ith least significant bit of the input if and only if the conjunction of the $i-1$ least significant bits of the input is equal to 1. Thus, the circuit comprises $n-1$ components (k-Toffoli gates), where the jth gate is an $(n-1-j)$-Toffoli gate that negates line $n-j$ and is controlled by the lines $1 \ldots n-2-j$; see Figure 3.

The nth line of the circuit in Figure 3 is required in order to realize the $(n-2)$-Toffoli gate. The line is used as a temporary register and retains its original value by the end of the computation of the $(n-2)$-Toffoli gate. Via straightforward induction on n, it is easy to see that the circuit realizes permutation π. Since the realization of each k-Toffoli gate comprises $O(k)$ normal Toffoli gates (2-Toffoli gates) [BBD$^+$95], the half incrementor may be realized in $O(n^2)$ gates. It follows that if we use an additional line, then a complete incrementor can be realized, otherwise, the best we can hope to realize is a nigh-incrementor.

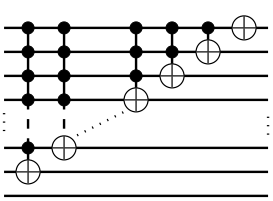

Figure 3. A half-incrementor circuit.

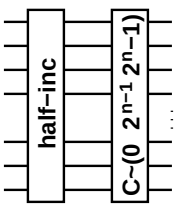

Figure 4. A nigh-incrementor circuit.

The nigh-incrementor is realized by concatenating an additional circuit onto the one that realizes a half-incrementor. Since the nigh-incrementor corresponds to the permutation π', let $\rho = \pi^{-1}\pi' = (0\ 2^{n-1}\ 2^n - 1)$, and thus the circuit $C_\pi C_\rho = C_{\pi'}$, depicted in Figure 4, realizes the nigh-incrementor. By Corollary 6, the complexity of C_ρ is $O(n)$. Hence, the circuit complexity of the nigh-incrementor is also $O(n^2)$. The half-incrementor is also the basic component in the construction of the adder.

In contrast to the incrementor, the adder requires no additional lines; this is because the adder corresponds to an even parity permutation. An adder that takes two n-bit inputs, on $2n$ lines and outputs the result on the latter n lines, $n+1,\ldots,2n$, and the first summand on the former n lines, $1,\ldots,n$. The adder comprises a sequence of n controlled half-incrementors; see Figure 5. The kth half-incrementor is controlled by line k, $k \in [n]$, and increments the $n-k$ most significant lines of the second summand, i.e., the increment is performed on lines $n+k,\ldots,2n$. This follows from the observation, that adding 2^j to an n-bit value corresponds to performing an increment on the $n-j$ most significant bits. The adder does exactly that, performing a controlled increment for each of the n bits of the first summand. Since each half-incrementor can be realized in $O(n^2)$ gates, the entire adder can be realized in $O(n^3)$ gates.

Unfortunately, there seems little that can be done to reduce this bound. For example, implementing a ripple adder is difficult because each stage of the ripple adder loses information—the preceding carry—implying that in order for a ripple adder circuit to work reversibly, all carry information needs to saved; we know of no way to accomplish this.

Realization of Threshold Functions. Quantum computing is inherently a probabilistic model of computation. Therefore, threshold functions, including the consensus function, are themselves necessarily useful in quantum computing. Except for one case—a majority of an odd number of variables—none of the threshold functions have an n-embedding. However all threshold functions have an $n+1$ embedding, and in many cases an input-preserving one. The threshold functions that are easiest to realize are the consensus, conjunction, and disjunction functions on n variables.

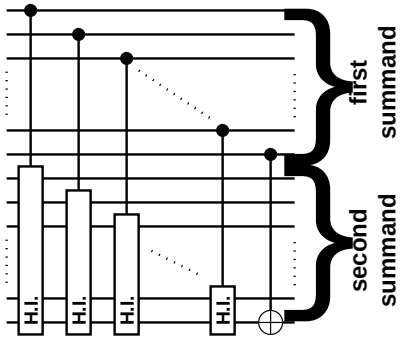

Figure 5. An adder circuit.

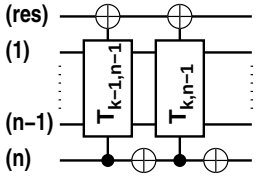

Figure 6. A $T_{k,n}$ circuit.

The consensus function evaluates to 1 if and only if all n variables have the same value. In fact, this function has input-preserving $(n+1)$-embedding comprising two transpositions: $(0\,2^n)(2^n-1\;2^{n+1}-1)$. Thus, by Corollary 7, consensus has a concise realization of size $O(n)$. On the other hand, the conjunction function—as well as its dual, disjunction—do not have an input-preserving $(n+1)$-embedding because the embedding would be an odd parity permutation, comprising one transposition: $(2^n-1\;2^{n+1}-1)$. However, there are many nearly input-preserving embeddings, like $(1\,2)(2^n-1\;2^{n+1}-1)$, whose complexity, by Corollary 7, is also $O(n)$.

More complicated threshold functions can be realized by composing the $\sum_{i=k}^{n}\binom{n}{k}$ transpositions, (xx'), where $|x|\geq k$ and $x'=2^n\oplus x$; each transposition corresponds to a 1 in the threshold's truth table. We assume that $k\geq n/2$ since computing the dual only requires an additional $O(n)$ gates. This yields realizations of size $O\left(\binom{n}{k}n\right)$ for threshold function $T_{k,n}$, $k\geq n/2$, where n is the number of variables and k is the threshold. However, the resulting realization are not necessarily obvious. We present a recursive construction that yields realizations with the same asymptotic complexity, but with a more analyzable structure.

A realization of threshold function $T_{k,n}$ comprises two simpler threshold function realizations. The two components of a realization of $T_{k,n}$ are a controlled realization of $T_{k-1,n-1}$, and a controlled realization of $T_{k,n-1}$; see Figure 6.

If line n has value 1, then the circuit needs only to check that the weight of the remaining $n-1$ lines is $k-1$ or greater. The first controlled component, which realizes $T_{k-1,n-1}$, performs this function. Otherwise, if line n has value 0, the weight of the remaining $n-1$ lines must be k or greater if the threshold is to be met. The second controlled component—a $T_{k,n-1}$ that is controlled by the negation of line n—performs this task. Each of the components are realized in the same way; the base cases, $T_{1,m}$ and $T_{m-1,m}$, are simply disjunctions and conjunctions over m variables, where $1\leq m\leq n$.

The complexity of this construction, particularly for the majority function, is exponential in n. The recurrence relation $R(k,n)=R(k-1,n-1)+R(k,n-1)$ describes the complexity of the construction—each of the two terms includes one of the two additional NOT gates. The boundary conditions are $R(1,m)=R(m,m)=cm$, where c is constant factor. Thus, the complexity of the realization of $T_{k,n}$, is $O\left(\binom{n}{k}k\right)$. Not surprisingly, the complexity of the realization is the same for both constructions. The threshold function is a prime example characterizing the application of Corollaries 6 and 7. Namely, that many Boolean functions have recursive realizations. The corollaries are useful for creating realizations of the base cases, which are then composed to yield the entire realization.

5. Techniques for Realizing Reversible Circuit

Three repeatedly used mechanisms for realizing circuits are commutators, conjugates, and "don't cares". The commutator of two circuits $[C_\sigma,C_\tau]=C_\sigma C_\tau C_\sigma^{-1}C_\tau^{-1}$—assuming that permutations σ and τ do not commute—is a mechanism for combining two circuits, which are controlled by distinct sets of control lines, into one that is controlled by union of the control lines, e.g., Barenco et al. [BBD$^+$95] used this approach to construct $(n-2)$-Toffoli gates.

The conjugate of one circuit by another, $C_\sigma C_\tau C_\sigma^{-1}$, preserves the cycle structure realized by C_τ, but changes the points within the cycles. This mechanism is useful for massaging a circuit that does 'almost the right thing' into one that performs the required permutation. Conjugation was heavily used in the proof of the main result, particularly in the construction and transformation of 3-cycles. Conjugation decouples circuit structure from input representation, i.e., if the structure of the permutation that is realized by the circuit is correct, then the circuit can easily be adapted to work on the right set of inputs with a small amount of additional circuitry.

An input-preserving realization of the conjunction function corresponds to a singe transposition—an odd parity permutation. Since an odd parity permutation cannot be realized by a circuit on four or more wires [CG75], an even permutation, comprising two transpositions, is used. The additional permutation affects two other points of the input but does not affect the output; namely, we sacrifice the input-preserving property to achieve the realization. In a sense we take advantage of the fact that we "don't care" what the outputs of the input carrying lines is, provided that the line carrying the output value is correct. This approach is similar to the Karnaugh-maps method [Kar53], which is used for optimizing general combinational circuits.

6. Discussion

Reversible circuits are a concrete model of reversible computation that also satisfy the underpinnings of quantum computation; namely, quantum computation must be reversible. Since classical Boolean functions will necessarily comprise some building blocks of a quantum computer, determining how these functions can be reversibly realized is an important problem. We have shown that if a Boolean function f can be embedded into a permutation σ on an n-dimensional Boolean cube, such that the cycle representation of σ is of size s, then function f can be realized by a reversible circuit of size $O(sn)$. Furthermore, we showed how these results can be applied by detailing realizations of several families of Boolean functions such as incrementors and threshold functions.

One of the motivations of this work is quantum computation. One can ask how quantum circuits—which were introduced by Feynman [Fey86] and formalized by Deutch [Deu89]—compare to reversible circuit. In 1995, Barenco et al. [BBD$^+$95] showed that quantum circuits can realize all permutations on an n-line circuit. Thus, not only are quantum circuits strictly more powerful than reversible circuits, they can also be more concise. However, the issue of whether quantum circuits are exponentially more powerful than reversible circuits remains open. Although, the famous factoring algorithm of Shor [Sho97], may indicate an affirmative answer, the result of Valiant's [Val01] indicates that in many cases the answer is negative. Since quantum circuits can simulate reversible circuits with no overhead—the Toffoli and NOT gates are commonly included in the basic set of quantum gates [BBD$^+$95]—the results in this paper are also applicable in the quantum setting.

There is a useful analogy between our result and the fact that functions with heavily unbalanced truth tables have concise circuit realizations. Namely, if the ratio of 1s to 0s in the truth table is $O(2^{-n}n^c)$ or $\Omega(2^n n^{-c})$ for some constant c, then the number of terms in the corresponding disjunction—and the number of transpositions realized by

the corresponding reversible circuit—is small. However, a function whose truth table is relatively balanced may also have a small realization.

Unfortunately, if a Boolean function f is nearly balanced and has an embedding whose cycle representation is exponentially large, there is no way to determine if f has a polynomial-size reversible circuit representation. For example, an n-line circuit comprising a single NOT gate realizes a permutation whose cycle representation comprises 2^{n-1} disjoint transpositions!

One possible approach is to partition the transpositions into equivalence classes based upon the behaviour of the transposition on a lower order Boolean cube. For example, a NOT gate on the first line of a circuit performs the permutation $(0\,1)$ on a one line circuit, $(0\,1)(2\,3)$ on a two line circuit, and $\prod_{i=0} 2^{n-1}(2i\ 2i+1)$ on an n-line circuit; all belong to the same equivalence class. Put another way, if a permutation can be projected onto a lower dimensional Boolean cube, and the sub-cube can be embedded into the original Boolean cube to yield the original permutation, then both the projection and original permutation belong to the same equivalence class. The complexity of realizing any element of the class is equal to the complexity of realizing the smallest element. In the example above, the NOT equivalence class has a complexity of 1, regardless of n. If a permutation can be factored into representatives of equivalence classes, then the complexity of realizing the permutation is simply the sum over the complexities of each of the representatives.

Even this is insufficient, because the nigh-incrementor, has a cycle representation that comprises a single exponentially large cycle. Yet, as we have shown, the nigh-incrementor has a concise realization. Unlike in the preceding case, projecting the permutation onto a lower dimensional Boolean cube does not work. Factoring a permutation into representatives is itself a difficult problem. For example, the decomposition of the nigh-incrementor into a 3-cycle and a half-incrementor is not at all obvious without a priori knowledge.

Finally, we note that a reversible realization can easily be realized by an irreversible circuit whose size and depth is only a constant factor larger than the reversible realization. In essence, a bound on the reversible complexity of a Boolean function automatically yields a bound on the classical circuit complexity of the function; the converse, is not true [LV96, LTV98, BTV01]. Thus, determining if a Boolean function has a concise reversible circuit realization remains an open an challenging problem. In fact, a simpler question should be answered first: can the n-adic majority function be efficiently realized by an $(n+c)$-line reversible circuit, where c is a constant? Alternatively, either improving the realization of the half-incrementor or proving a quadratic lower bound would also be of great interest.

References

[Bar86] D. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in $\mathbf{NC}^1$. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 1–5, 1986.

[BBD+95] A. Barenco, C. Bennett, D. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter. Quantum gates and circuits. *Phys. Rev. A.*, 52:3457–3467, 1995.

[Ben73] C. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17:198–200, 1973.

[Ben89] C. Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, 1989.

[BTV01] H. Buhrman, J. Tromp, and P. Vitányi. Time and space bounds for reversible simulation. In *arXiv:quant-ph/0101133*, 2001.

[CG75] D. Coppersmith and E. Grossman. Generators for certain alternating groups with applications to cryptogaphy. *SIAM Journal on Applied Mathematics*, 29(4):624–627, 1975.

[Cle90] R. Cleve. Complexity theoretic issues concerning block ciphers related to D.E.S. In A. J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology—CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 530–544. Springer-Verlag, 1990.

[Deu89] D. Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London, Series A*, 425:73–90, 1989.

[EG83] S. Even and O. Goldreich. DES-like functions can generate the alternating group. *IEEE Trans. on Information Theory*, 29(6):863–865, 1983.

[Fey86] F. Feynman. Quantum mechanical computers. *Foundations of Physics*, 16(6):507–531, 1986.

[FT82] E. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3/4):219–253, 1982.

[Kar53] M. Karnaugh. The map method for synthesis of combinational logic circuits. *AIEE Transactions, Part I Communication and Electronics*, 72:593–599, 1953.

[Lan61] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5:183–191, 1961.

[LMT00] K. Lange, P. McKenzie, and A. Tapp. Reversible space equals deterministic space. *Journal of Computer and System Sciences*, 60(2):354–367, 2000.

[LS90] R. Levine and A. Sherman. A note on Bennett's time-space tradeoff for reversible computation. *SIAM Journal on Computing*, 19(4):673–677, 1990.

[LTV98] M. Li, J. Tromp, and P. Vitányi. Reversible simulation of irreversible computation. *Physica D*, 120:168–176, 1998.

[LV96] M. Li and P. Vitányi. Reversible simulation of irreversible computation. In *Proceedings of the 11th IEEE Computational Complexity Conference*, pages 306–306, 1996. Submitted to Physica D, 1997.

[Max71] J. Maxwell. *Theory of Heat*. Longmans, Green and Co., London, 1871.

[Sha48] C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[Sho97] P. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[SW49] C. Shannon and W. Weaver. *A Mathematical Theory of Communication*. University of Illinois Press, Urbana, Illinois, 1949.

[Szi29] L. Szilard. Über die Entropieverminderung in einem thermodynamischen System bei eingriffen intelligenter wesen. *Zeitschrift für Physik*, 53:829–856, 1929.

[Tof80] T. Toffoli. Reversible computing. In *Automata, Languages and Programming, 7th Colloquium*, volume 85 of *Lecture Notes in Computer Science*, pages 632–644, 1980.

[Val01] L. Valiant. Quantum computers that can be simulated classically in polynomial time. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 114–123, 2001.