Reversible Logic Circuit Synthesis [*]

Vivek V. Shende, Aditya K. Prasad, Igor L. Markov, and John P. Hayes
University of Michigan, Advanced Computer Architecture Laboratory, Ann Arbor, MI 48109-2122
{vshende,akprasad,imarkov,jhayes}@umich.edu

ABSTRACT

Reversible, or information-lossless, circuits have applications in digital signal processing, communication, computer graphics and cryptography. They are also a fundamental requirement for quantum computation. We investigate the synthesis of reversible circuits that employ a minimum number of gates and contain no redundant input-output line-pairs (temporary storage channels). We propose new constructions for reversible circuits composed of NOT, Controlled-NOT, and TOFFOLI gates (the *CNT* gate library) based on permutation theory. A new algorithm is given to synthesize optimal reversible circuits using an arbitrary gate library, and we describe much faster heuristic algorithms. We also pursue applications of the proposed techniques to the synthesis of quantum circuits.

1. INTRODUCTION

In most computing tasks, the number of output bits is relatively small compared to the number of input bits. For example, in a decision problem, the output is only one bit (yes or no), and the input can be as large as desired. However, computational tasks in digital signal processing, communication, computer graphics and cryptography require that all of the information encoded in the input be preserved in the output. Some of those tasks are important enough to justify new microprocessor instructions to HP PA-RISC (MAX and MAX-2), Sun SPARC (VIS), PowerPC (AltiVec), IA-32 and IA-64 (MMX) instruction sets [15, 9]. In particular, new bit-permutation instructions were shown to vastly improve performance of several standard algorithms, including matrix transposition and DES, as well as recent cryptographic algorithms Twofish and Serpent [9]. Bit-permutations are a special case of *reversible functions*, that is,

functions that permute the set of possible input values. For example, the butterfly operation $(a, b) \rightarrow (a+b, a-b)$ is reversible but isn't a bit permutation. It is a key element of Fast Fourier Transform algorithms and has been used in application-specific processors from Tensilica. One might expect to get further speed-ups by adding instructions to allow computation of an arbitrary reversible function; the problem of chaining such instructions together provides one motivation for studying reversible logic circuits, that is, logic circuits composed of gates computing reversible functions.

Reversible circuits are also interesting because the loss of bits of information implies energy loss [2]. Younis and Knight [18] showed that some reversible circuits can be made asymptotically energy-lossless if their delay is allowed to be arbitrarily large. Currently, energy losses due to irreversibility are dwarfed by the overall power dissipation, but this may change if Moore's law holds until 2020 and power dissipation improves [11]. In particular, reversibility is important for nanotechnologies where switching devices with gain are difficult to build.

Finally, reversible circuits can be viewed as a special case of quantum circuits because quantum evolution must be reversible. Classical (non-quantum) reversible gates are subject to the same "circuit rules", whether they operate on classical bits or quantum states. In fact, popular universal gate libraries for quantum computation often contain, as their subsets, universal gate libraries for classical reversible computation. While the speed-ups which make quantum computing attractive are not available without purely quantum gates, logic synthesis for classical reversible circuits is a first step toward synthesis of quantum circuits. Moreover, algorithms for quantum communications and cryptography often do not have classical competitors because they act on quantum states, even if their action in a given computational basis corresponds to classical reversible functions on bit-strings. Another connection between classical and quantum computing comes from "pseudo-classical" circuits, as used, e.g., in Grover's search algorithm [4]. These circuits are close to classical reversible circuits [5] and their definition involves an arbitrary one-output (irreversible) Boolean function.

We now briefly review existing work on classical reversible circuits. Toffoli [16] gives constructions for an arbitrary reversible or irreversible function in terms of a certain gate library. However, his method makes use of a large (although linear in the input size) number of temporary storage channels, i.e. input-output wire-pairs other than those on which the function is computed. Sasao and Kinoshita show that any conservative function ($f(x)$ is conservative if $\forall x$, x and $f(x)$ contain the same number of 1s in their binary expansions) has an implementation with only 3 temporary storage channels using a certain fixed library of conservative gates, although no explicit construction was given [12]. Kerntopf uses exhaustive search methods to examine small scale synthesis problems

and related theoretical questions about reversible circuit synthesis [6]. Finally, members of the Portland Quantum Logic Group propose a general heuristic for reversible logic synthesis [11].

Our work pursues synthesis of optimal reversible circuits which can be implemented without temporary storage channels. In Section 3 we show by explicit construction that any reversible function which performs an even permutation on the input values may be synthesized using the *CNT* (CNOT, NOT, TOFFOLI) gate library under such constraints. In Section 4 we present synthesis algorithms for decomposing such a function into a circuit with a minimal number of gates. Besides branch-and-bound, we use dynamic programming that exploits reversibility. Empirical results are given in Section 5, and applications to quantum computing in Section 6.

2. BACKGROUND

In conventional (irreversible) circuit synthesis, one typically starts with a universal gate library and some specification of a Boolean function. The goal is to find a logic circuit that implements the Boolean function and minimizes a given cost metric, e.g., the number of gates or the circuit depth. At a high level, reversible circuit synthesis is just a special case in which no fanout is allowed and all gates must be reversible.

DEFINITION 1. *A gate is reversible if the (Boolean) function it computes is bijective.*

A necessary condition is that the gate have the same number of input and output wires. If it has k, it is called a $k \times k$ gate, or a gate on k wires. We will think of the mth input wire and the mth output wire as really being the same wire. Many gates satisfying these conditions have been examined. We will consider a specific set, defined by Toffoli [16].

DEFINITION 2. *A k-CNOT is a $(k+1) \times (k+1)$ gate. It passes the first k inputs through unchanged, and inverts the last iff the others are all 1. (These gates are all reversible.)*

The first three of these have special names. The 0-CNOT is just an inverter, referred to as a NOT gate. The 1-CNOT, which passes the first input through and inverts the second if the first input is 1, is referred to as a CNOT (controlled-NOT). The 2-CNOT is called a TOFFOLI gate. Together, the NOT, CNOT, and TOFFOLI form a universal set of gates for classical reversible circuits [16] (we will be more specific about what this means later). They are also attractive for quantum computing [10] where additional, purely quantum, gates are required for universality.

DEFINITION 3. *A well-formed reversible logic circuit is an acyclic combinational logic circuit in which all gates are reversible, and are interconnected without fanout.*
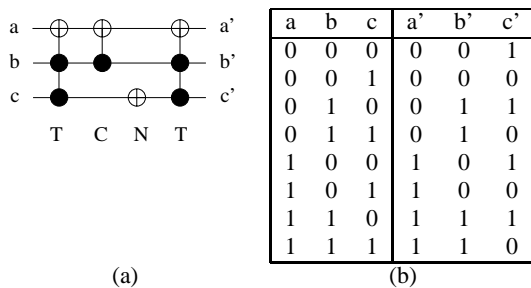

a	b	c	a'	b'	c'
0	0	0	0	0	1
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	0	1
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	1	0

(a)   (b)

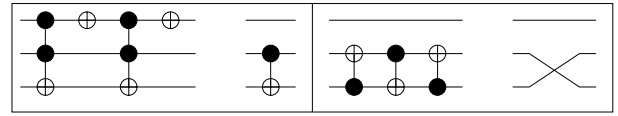**Figure 1: (a) Reversible circuit, (b) the function it implements.**

Figure 2: Two reversible circuit equivalences. $T(1,2;3) \cdot N(1) \cdot T(1,2;3) \cdot N(1) = C(2;3)$**, and** $C(3;2) \cdot C(2;3) \cdot C(3;2) = S(2,3)$

We will be working with circuits from a pre-given, limited gate library. Usually, this will be the *CNT* gate library, consisting of the CNOT, NOT, and TOFFOLI gates defined above. Sometimes, we will take subsets, and speak of, say, the *CT* gate library. We will also on occasion add the SWAP gate, *S*, a 2×2 gate which exchanges the inputs; that is, $(x,y) \rightarrow (y,x)$.

As with reversible gates, a reversible circuit has the same number of input and output wires; again we will call a reversible circuit with n inputs an $n \times n$ circuit, or a circuit on n wires. We may also draw an "invisible box" around a $n \times n$ circuit and think of it as the inner workings of an $n \times n$ reversible gate.

This also allows us to draw reversible circuits as vertical arrays of horizontal lines (representing wires), in which gates are represented by vertically oriented symbols. For example, in Figure 1, we see a reversible circuit drawn in standard notation [10]. The $\oplus$ symbols represent inverters and the $\bullet$ symbols represent controls. A vertical line connecting a control to an inverter means that the inverter is only applied if the wire on which the control is set carries a signal. Thus, the gates used are (from left to right), TOFFOLI, CNOT, NOT, TOFFOLI.

The truth table in Figure 1.b is the only truth table appearing in this paper. Since we will be dealing only with bijective functions from $B^k \rightarrow B^k$, we represent them using the *cycle notation*, known from elementary algebra. That way, every permutation is represented by disjoint cycles of variables. For example, the truth table in Figure 1 is represented by $(0,1)(2,3)(4,5)(6,7)$ because the corresponding function swaps 000 (0) and 001 (1), 010 (2) and 011 (3), etc. The set of all permutation of n letters is denoted S_n, so the set of bijective functions from B^n to itself is S_{2^n}.

As the permutation $(0,1)(2,3)(4,5)(6,7)$ may be computed in a circuit with only gates from the *CNT* gate library, we will call it *CNT*-constructible. More generally:

DEFINITION 4. *Let L be a (reversible) gate library. An L-circuit is a circuit with only gates from L, and a permutation $\pi \in S_{2^n}$ is L-constructible if it may be computed by an $n \times n$ L-circuit.*

Note that a circuit consisting of just an inverter on the *c-c'* line computes the same function as the circuit in Figure 1. Pairs of circuits computing the same function are very useful, since we may substitute one for another. Two more such pairs are given in Figure 2. On the right, we see that three *C* gates may be used to replace a *S* gate; on the left we see that the *CNT* gate library is redundant, in that we may replace every occurrence of a *C* gate with two *T* gates and two *N* gates. We will still use the *CNT* gate library in synthesis to reduce gate counts and potentially speed up synthesis (Figure 2, shows how to replace 4 gates with one *C* gate). In fact, for the same reason, we will sometimes add the *S* gate, and consider the *CNTS* gate library.

DEFINITION 5. *Two reversible circuits are equivalent if they compute the same function.*

The left box in Figure 2 illustrates the use of "temporary storage". Computing a CNOT usually only takes 2 wires, but if we do

it with two NOT gates and two TOFFOLI gates, we need a third wire. The value of the third wire is irrelevant to the computation, and emerges unaltered. More generally, consider the general reversible circuit of Figure 3. The top $n-k$ lines transfer $n-k$ signals Y to the corresponding wires on the other side of the circuit. The bottom k go in as the input value X and emerge as the output value $f(X)$. These lines often serve as an essential workspace for computing $f(X)$. Following Toffoli, we say that C computes $f(X)$ using $n-k$ lines of temporary storage [16]. Figure 1 provides another example; the circuit there computes NOT x on the c-c' wire, using the top two wires as temporary storage.

We now formally define what it means for a library L of reversible gates to be *universal*.

DEFINITION 6. *Let L be a reversible gate library. Then L is universal if for all permutations $\pi \in S_{2^k}$ (for all k) there exists some l such that some L-constructible circuit computes π using l wires of temporary storage.*[1]

3. THEORETICAL RESULTS AND HEURISTICS FOR SYNTHESIS

It is a result of Toffoli that the *CNT* gate library is universal; he also showed that one may bound the amount of temporary storage required to compute a permutation in S_{2^n} by $n-3$. We are interested in trying to synthesize permutations using no extra storage. As a first step, we would like to know for which permutations this may actually be done, using a reasonable gate library. Toffoli gave a negative result in this direction, but to state it we must introduce the concept of an even permutation.

DEFINITION 7. *A permutation is called even if it may be written as the product of an even number of transpositions.*[2] *The set of even permutations in S_n is denoted A_n, and it is a result of elementary algebra that half the permutations are even for $n > 1$.*

PROPOSITION 1. *Any $n \times n$ circuit with no $n \times n$ gates computes an even permutation* [16].

In particular, since the *CNT* gate library contains no gates of size greater than 3, Proposition 1 implies that every *CNT*-constructible permutation is even for $n \geq 4$. We now investigate the converse.

PROPOSITION 2. *Every even permutation is CNT-constructible.*

Proof: It follows from a result of Toffoli [16] that every permutation in S_{2^n} is *CNT*-constructible for $n < 4$. Suppose $n \geq 4$. Any

permutation $\pi \in A_{2^n}$ may be written as the product of disjoint transposition pairs. For a proof, consult Proposition 12 in the appendix. It therefore suffices to show that disjoint transposition pairs are constructible, as we may chain together their circuits to obtain the circuit for π. First, we observe that the permutation with cycle decomposition $(0,1)(2,3)$ can be computed by a circuit consisting of a $(n-2)$-CNOT gate with the controls on the top $n-2$ wires and the inverter on the bottom wire, with an N gate on each side of each control. We may replace the $(n-2)$-CNOT gate with a linear (in n) number of C gates [1].

Let $S = \{a, b, c, d\}$ with a, b, c, d distinct. In Proposition 13 in the appendix, we explicitly construct a circuit computing a permutation π_S such that $\pi_S(a) = 0$, $\pi_S(b) = 1$, $\pi_S(c) = 2$, and $\pi_S(d) = 3$. Because $(a,b)(c,d) = \pi_S(0,1)(2,3)\pi_S^{-1}$, there is a circuit computing $(a,b)(c,d)$ by chaining together first a circuit computing π_S, then the circuit computing $(0,1)(2,3)$, and finally a circuit computing π_S^{-1}.

Finally, if $c(\pi)$ is the cardinality of the support of π, then there are $\Theta(nc(\pi))$ gates used in this construction. $c(\pi) < 2^n$. $\square$

The following two corollaries (i) give a way to synthesize circuits computing odd permutations using temporary storage, and (ii) extend the result of Proposition 2 to an arbitrary universal gate library.

PROPOSITION 3. *If $\pi \in S_{2^n}$ is any permutation, then we may compute π using the CNT gates and one wire of temporary storage.*

Proof: Suppose we had a $n \times n$ gate G computing π, and we placed it on the bottom n wires of an $(n+1) \times (n+1)$ reversible circuit; let $\bar{\pi}$ be the permutation computed by this new circuit. Then by Proposition 1, $\bar{\pi}$ is even, so by Proposition 2, $\bar{\pi}$ is *CNT*-constructible. Let C be a *CNT*-circuit computing $\bar{\pi}$. Then C computes π with one line of temporary storage. $\square$

PROPOSITION 4. *If L is any universal gate library, then for sufficiently large n, permutations in A_{2^n} are L-constructible, and permutations in S_{2^n} are computable with one wire of temporary storage.*

Proof: Since L is universal, there is some number k such that we may compute the permutations corresponding to the NOT, CNOT, and TOFFOLI gates using at most k total wires. Let $n > k$, and let $\pi \in A_{2^n}$. By Proposition 2, we may find a *CNT*-circuit C computing π; replace every occurrence of N, C, or T with a circuit computing it. The second claim follows similarly from Propositions 2, 3. $\square$

Since Proposition 2 is proven by an explicit construction, we may implement it as a circuit synthesis heuristic which produces (very) suboptimal circuits. For permutations in A_{2^n}, the runtime and the length of the circuits produced are both $O(n2^n)$, suggesting that this technique should work with circuits of up to approximately 20 inputs.

Later, we describe an algorithm which will synthesize optimal circuits from an arbitrary gate library. Roughly speaking, the performance of this algorithm is improved by using a smaller gate library, as long as the average circuit length is not significantly increased. We will show that the inverters in a *CNT*-circuit may be pushed to the end of the circuit.[3]

DEFINITION 8. *If $L_1 \dots L_k$ are gate libraries, an $L_1 | \dots | L_k$-circuit is an L_1-circuit followed by a L_2-circuit, $\dots$, followed by an L_k-circuit. A permutation computed by an $L_1 | \dots | L_k$ circuit is "$L_1 | \dots | L_k$-constructible".*

---

[1] Note that we do not allow constant signals.

[2] It is a result from elementary algebra that if a permutation may be written as the product of an odd number of transpositions, then it may not be written as an even number of transpositions.
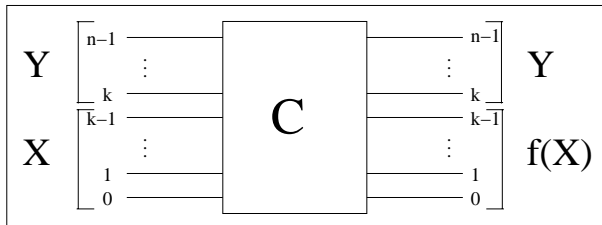

Figure 3: A reversible circuit with n-k wires of temp. storage.

[3] This is analogous to pushing all inverters in an AND-OR-NOT circuit to the inputs by applying De Morgan's laws.
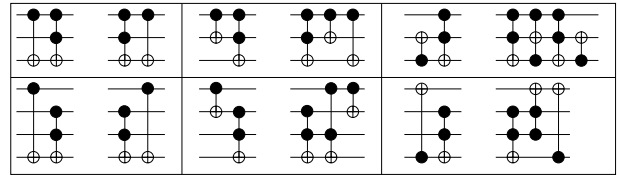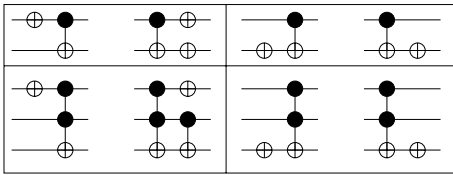
Figure 4: Equivalences between reversible circuits used in our constructions.

PROPOSITION 5. *Every CNT-circuit is equivalent to some CT|N-circuit.*

Proof: First, we move all the N gates toward the outputs of the circuit. Each box in Figure 4-left indicates a way of replacing an $N|CT$ circuit with a $CT|N$ circuit. Moreover, every possible way for an N gate to appear to the immediate left of a C or a T is accounted for, up to permuting the input and output wires.

Now, number the non-N gates in the circuit in a reverse topological order starting from the outputs. In particular, if two gates appear at the same level in a circuit diagram, they must be independent, and one can order them arbitrarily. Let d be the number of the highest-numbered gate with an N gate to its left. All N gates past the d-th gate G may be reordered with the G gate without introducing new N gates on the other side of G. In any event, as there are no remaining N gates to the left of G anymore, d decreases. This process terminates with all the N gates are clustered together at the circuit outputs.

If we make sure to always cancel redundant N gates, then no more than 2 new gates will be introduced for each non-inverter originally in the circuit; additionally, there will be no more than n total N gates when the process is complete. Thus if the original circuit had l gates, then the new circuit has at most $3(l-1) + n$ gates. □

PROPOSITION 6. *The permutation π computed by a $CT|N$-circuit uniquely determines π_{CT} and π_N computed by the CT and N sub-circuits.*

Proof: C and T gates (and hence CT-circuits) fix 0. Thus $\pi(0) = \pi_N(0)$. But the image of 0 (or anything else) under an N-circuit completely determines the π_N. Then $\pi_{CT} = \pi\pi_N^{-1} = \pi\pi_N$. □

Thus, if we are looking for a CNT-circuit computing a permutation π, we may quickly compute π_N and then simplify the problem to that of looking for a CT-circuit computing $\pi\pi_N$. By Proposition 5, we know that the gate-minimal circuit of this form has at most about three times as many gates as the gate-minimal circuit computing π.

Given that the pictures in Figure 4-right show how to move a C gate past a T gate, and account for every possible way a C may appear to the left of a T (up to permuting wires), one might expect every CT circuit to be equivalent to a $T|C$ circuit. This is not the case. We note that the proof of Proposition 5 in fact requires the ability to move an arbitrary number of N gates past any other given gate, while Figure 4-right only allows us to move 1 C gate past a given T gate. However, many CT circuits are equivalent to $T|C$ circuits, and in this case the following result holds:

PROPOSITION 7. *The permutation π computed by a $T|C$-circuit uniquely determines permutations π_T and π_C computed by the T and C sub-circuits.*

Proof: Any C-circuit is linear [11], so it suffices to check its values on the basis elements corresponding to the binary expansions of 2^i.

As any T circuit fixes these, $\pi(2^i) = \pi_C \circ \pi_T(2^i) = \pi_C(2^i)$, so the permutation π uniquely determines π_C. $\pi_T = \pi\pi_C^{-1}$. □

Proposition 7 implies, in particular, that the number of permutations in S_{2^k} that are $T|C$-constructible is equal to the number that are C-constructible times the number that are T-constructible. In the results section, we will use this fact to show that there exist CT-constructible permutations which are not $T|C$-constructible.

4. OPTIMAL SYNTHESIS

Now that we know which permutations admit circuit realizations without extra storage, we seek *optimal* realizations of this type. A circuit is optimal if no equivalent circuit has smaller cost; in our case, the cost function will be the number of gates in the circuit.

PROPOSITION 8. *(Property of Optimality) If S is a sub-circuit of an optimal circuit C, then S is optimal.*

Proof: Suppose not. Then let S' be a circuit smaller (in the number of gates) than S, but computing the same function. If we replace S by S', we get another circuit C' which computes the same function as C. But since we have only modified S, we must have that C' is as much smaller than C as S' is smaller than S. However, C was assumed to be optimal, hence this is a contradiction. Note: there may be many equivalent circuits with the same number of gates as S. □

We will use an iterative-deepening A* (IDA*) search procedure [7] to find an optimal circuit computing a given permutation. An IDA* algorithm first examines possible solutions of cost 1, then possible solutions of cost 2, and so on. Pseudo-code for the IDA* framework of the algorithm is given in Figure 5. Once a circuit is found, it must be optimal since we have examined all smaller circuits already. We will use the property of optimality to speed up search.

```
CIRCUIT synthesize(PERM)
{
if (PERM==IDENTITY) return EMPTY_CCT;
// otherwise, use IDA* to find a circuit.
for(DEPTH ← 1, DEPTH < MAX_DEPTH; DEPTH++)
    {
    CIRCUIT ← find_circ(DEPTH, PERM, EMPTY_CCT);
    if (CIRCUIT != NIL) return CIRCUIT;
    }
}
```

Figure 5: Finding an optimal circuit computing permutation PERM. Returned value NIL means "not found". See `find_circ()` in Figure 6.

We know that the first k gates of an optimal circuit of cost n must form an optimal circuit. So, fixing k in advance, we extend the gate library into a "circuit library" of optimal circuits of cost k or

```
CIRCUIT find_circ(COST, PERM, CURR_CCT)
{
if (COST ≤ k)
   // if PERM can be computed by a circuit
   // with fewer at most k gates,
   // such a circuit must be in the library
   return CURR_CCT + LIB[DEPTH].find(PERM));
else
   // The goal circuit must have >k gates;
   // Try constructing it from k-gate circuits
   for each C in LIB[k]
   {
      // divide PERM by permutation computed by C
      PERM2 ← PERM * INVERSE(C.perm)
      // and try to synthesize the result
      TEMP_CCT ← find_circ(depth-k,PERM2));
      if (TEMP_CCT != NIL) return TEMP_CCT;
   }
}
```

Figure 6: Finding a circuit of cost ≤COST or less that computes permutation PERM (NIL returned if no such circuit exists). CURR_CCT, TEMP_CCT and records in LIB represent circuits, and include a field "perm" storing the permutation computed. The * character means concatenation of circuits, and NIL*anything=NIL.

less, and store them in an array LIB such that LIB[d] is a collection of d-gate circuits. Then, to find a circuit of cost n computing a given permutation, we iterate through optimal circuits of cost k, and for each we recursively look for an optimal circuit of cost $n-k$ that, together with the current circuit of cost k, computes the desired permutation. If $n-k$ is k or less, we look in our circuit library to check if the desired circuit exists. Pseudo-code is given in Figure 6. For any complete gate library IDA*-search terminates if and only if a circuit computing the desired function exists (the existence problem is addressed in Section 3). Therefore, our algorithm needs an additional termination condition that would prevent infinite looping if no solution exists. In general, we can stop the search procedure at some fixed cost, but if the total number of optimal circuits that may be synthesized with a given gate library is small enough to be stored in memory, we use a different approach. We store all permutations for which our algorithm finds circuits along the way. Suppose we discover that there are no optimal circuits of cost n (for some n). Then the property of optimality implies that any circuit with n gates or larger is suboptimal, and we may stop looking. Note that if a permutation is not synthesizable, this termination condition will trigger sooner or later. That is because there are only finitely many synthesizable permutations on w wires, versus infinitely many circuits on w wires.

Generating the circuit library may be done in the following fashion. We begin with a library of maximum cost 1, since this is just a gate library. To generate the library of maximum cost $k+1$ from a library with maximum cost k, it suffices to iterate through the optimal circuits of cost k and, for each, try each way of appending a gate to the end. By the property of optimality, this examines every optimal circuit of length $k+1$.

5. EMPIRICAL RESULTS FOR CLASSICAL REVERSIBLE CIRCUITS

We may use the algorithm explained in the previous section to find the length of the optimal circuit computing some given permutation. Doing this for all permutations which may be computed on, say, three wires, we can determine the distribution of optimal cir-

Size	N	C	T	NC	CT	NT	CNT	CNTS
12	0	0	0	0	0	47	0	0
11	0	0	0	0	0	1690	0	0
10	0	0	0	0	0	8363	0	0
9	0	0	0	0	0	12237	0	0
8	0	0	0	0	6	9339	577	32
7	0	0	0	14	386	5097	10253	6817
6	0	2	0	215	1688	2262	17049	17531
5	0	24	0	474	1784	870	8921	11194
4	0	60	5	393	845	296	2780	3752
3	1	51	9	187	261	88	625	844
2	3	24	6	51	60	24	102	134
1	3	6	3	9	9	6	12	15
0	1	1	1	1	1	1	1	1
Total	8	168	24	1344	5040	40320	40320	40320
Time, s	≈ 0	≈ 0	≈ 0	30	215	97	40	15

Table 1: Size distribution of optimal 3-wire L-circuits, for subsets L of the $CNTS$ gate library. Runtimes are given for 2GHz Pentium-4 Xeon.

cuit sizes. For example, Table 1 lists the number of L-constructible permutations for various subsets L of the $CNTS$ gate library.

While we cannot theoretically validate every entry of Table 1, we can check the totals. Every reversible function on 3 wires can be synthesized using the CNT gate library [16], and there are $8! = 40320$ of these. All those can be synthesized with the NT library because the C gate is redundant in the CNT library (Figure 2 shows how to replace a C gate with two N and two T gates). On the other hand, adding the S gate to the library cannot decrease the number of synthesizable functions. Therefore, the totals in the NT and $CNTS$ columns must be 40320 as well.

On the other side of the table, the number of possible N circuits is just $2^3 = 8$ since there are three wires, and there may be at most one N gate per wire in an optimal circuit (since otherwise we may cancel redundant pairs.) By Propositions 5 and 6, the number of NC-constructible permutations should be the product of the number of N-constructible permutations and the number of C constructible permutations, since any NC-constructible permutation may be written uniquely as a product of an N constructible permutation and a C constructible permutation. So the total in the NC column should be the product of the totals in the C and N columns, which it is. Similarly, the total in the CNT column should be the product of the totals in the CT and N columns; this would allow us to deduce the total number of CT-constructible permutations from values we know.

Finally, it is possible to show that the number of permutations implementable on n wires with C gates only is $\prod_{i=0}^{n-1}(2^n - 2^i)$. For $n = 3$ this formula gives 168 and agrees with Table 1. To derive this formula, observe that the C gate defines a linear transformation over the two-element field $\mathbb{F}_2$. This means that if we apply an n-wire circuit consisting of one C gate to two sets of input values, then the bit-wise $\oplus$ of the outputs equals the output of the circuit on the bit-wise $\oplus$ of the inputs. This leads us to consider $2^n \times 2^n$ matrices with 0-1 entries because they can capture linear operators over $\mathbb{F}_2$. Since C gates are invertible, we also require that all matrices be reversible. In algebra this matrix group is denoted $GL_n(\mathbb{F}_2)$ [11]. It turns out that the C-constructible permutations bijectively correspond to matrices in $GL_n(\mathbb{F}_2)$. To produce the counting formula above, observe that a linear mapping is fully defined by its values on basis vectors. There are $2^n - 1$ ways of mapping the 2^n-bit-string 10...0. Once we fixed its image, there are $2^n - 2$ ways of mapping 010...0, and so on. Each time we map one of these basis bit-strings it can't map to the subspace spanned by the previous

bit-strings. This is why we have $2^n - 2^i$ choices for the i-th basis bit-string. Once all basis bit-strings are mapped, the mapping of the rest is specified by linearity.

We observed that the longest optimal C-circuits on 3, 4 and 5 wires merely permute the wires. Our experimental data supports the conjecture that no optimal C-circuit on n wires has more than $3(n-1)$ gates, and the ones with $3(n-1)$ gates represent wire permutations that leave no wire fixed. However, an information-theoretic counting argument shows that the optimal gate count in an optimal C-circuit is at least $O(n^2/log(n))$. This asymptotic bound is produced by comparing the number of unique C-circuits on n wires and the number of circuits formed by chains of up to d C gates [13]. Since this non-constructive argument is based on counting, finding some worst-case circuits and describing families with worst-case asymptotics remains an interesting challenge.

Since wire-swaps require three gates from the CNT library, we tried adding the swap gate, S, to our gate library. On average, circuit sizes only improved by one gate. On a 2GHz Pentium-4 Xeon, if we do not make use of a circuit library generating the Table 1 is not possible in many hours. Generating a circuit library up to three gates ($k = 3$) takes less than a minute, and all of Table 1 can be generated in minutes.

Although it is unrealistic to produce complete statistics for 4-wire functions (there are 16! of them), average synthesis times for takes less than a second when the input function can be implemented in 8 gates or less (in this case, our circuit library contains optimal circuits with up to 4 gates). CNT-constructible functions requiring 9 or more gates have been observed to take at least 1.5 hours to synthesize. The reason for the large jump after the 8 gate mark is that this is when the recursion starts having to go 3 levels deep. Improving the way the circuit library is stored would improve performance, but, it is unrealistic to expect optimal synthesis methods to scale very far. This algorithm does, however, scale better than its irreversible counterparts [8], primarily because application of the property of optimality to the reversible case is much more straightforward.

6. QUANTUM COMPUTING APPLICATIONS

This section presumes some familiarity with quantum computing not required elsewhere in the paper. Background can be found in the book [10] by Nielsen and Chuang. Grover's search algorithm is a quantum algorithm that allows one to search N unordered items in $O(\sqrt{N})$ time, where the desired items are identified by a Boolean predicate. This improves upon the best possible asymptotics for classical computation, which is $O(N)$.

Grover's search uses a quantum circuit that flips the sign of those states in the computational basis that satisfy the predicate, and leaves all other states unchanged. Quantum states in the computational basis can be thought of as strings of N bits. If f is a Boolean function that evaluates to 1 on desired basis states and 0 on other basis states, the circuit needs to change an arbitrary basis state $|x\rangle$ to the state $(-1)^{f(x)}|x\rangle$. Such a circuit computes a transformation given by a diagonal matrix with entries ± 1 and can be synthesized from basic gates for quantum computation [5].

Most works on Grover's algorithm do not address the synthesis of the above quantum circuits defined by Boolean functions f. According to Bettelli et al. [3], this is a major obstacle for automatic compilation of C++-like quantum programs, and no solutions are known.

PROPOSITION 9. *The problem of synthesizing a quantum circuit that transforms computational basis states $|x\rangle$ to $(-1)^{f(x)}|x\rangle$ can be reduced to a problem in the synthesis of classical reversible*

Circuit Size	0	1	2	3	4	5	6	7	Total
#circuits of that size	1	7	21	35	35	24	4	1	128

Table 3: Optimal 3+1 oracle circuits for Grover's search algorithm.

circuits [5].
Proof: Define the permutation $\hat{f}$ by $\hat{f}(x,y) = (x, y \oplus f(x))$, and define a unitary operator U_f by letting it permute the states of the computational basis according to $\hat{f}$. An additional qubit (wire) is initialized to the state $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ so that $U_f|x,-\rangle = (-1)^{f(x)}|x,-\rangle$ If we now ignore the value of the last qubit, the system is in the state $(-1)^{f(x)}|x\rangle$, which is exactly the state needed for Grover's algorithm. Since a quantum operator is completely determined by its behavior on a given computational basis, any circuit implementing $\hat{f}$ implements U_f. In particular, since reversible gates may be implemented with quantum technology, we may synthesize U_f as a reversible logic circuit. □

We are interested in circuits for such functions, therefore we first question whether $\hat{f}$ is CNT-constructible, i.e., whether the permutation $\hat{f}$ is even. Since $\hat{f}$ swaps (x,y) with $(x, y \oplus f(x))$, it may be written as a product of $\#\{x : f(x) = 1\}$ transpositions. Thus $\hat{f}$ is even iff this set has even cardinality, which is true for 50% of functions f.

Given a CNT-constructible $\hat{f}$, we can use the algorithm given in Section 4 to find the smallest circuit taking this form. Figure 3 gives the optimal circuit sizes of functions $\hat{f}$ corresponding to 3-input 1-output functions f (we call such functions "3+1 oracles" because they operate on four wires). These circuits are significantly smaller than many optimal circuits on four wires. This is not surprising, as they perform less computation.

In Grover oracle circuits the main input lines preserve their input values and only the temporary storage lines may change their values. Therefore Travaglione et al. [17] studied circuits where some lines cannot be changed even at intermediate stages of computation. In their terminology, a circuit with k lines that we are allowed to modify and an arbitrary number of read-only lines is called a k-bit ROM-based circuit. They show how to compute permutation $\hat{f}$ arising from a Boolean function f using a 1-bit quantum ROM-based circuit, and prove that if only classical gates are allowed, two writable bits are necessary. Two bits are sufficient if the CNT gate library is used. Their synthesis algorithms rely on EXOR-SUM decompositions of f. We state their result and outline their construction.

PROPOSITION 10. *Given an EXOR-SUM decomposition of a function f, we may synthesize a reversible 2-bit ROM based CNT-circuit computing $(x,a,b) \rightarrow (x,a,b \oplus f(x))$, where x is a k bit input [17].*
Proof: It suffices to know how to transform $(x,a,b) \rightarrow (x,a,b \oplus p)$ for an arbitrary product of uncomplemented literals p; because then we may add the terms in an EXOR-SUM decomposition term by term. So, without the loss of generality, let $p = x_1 \ldots x_m$.

Denote by $T(a,b;c)$ a T gate with controls on a,b and inverter on c. Similarly, denote by $C(a;b)$ a C gate with control on a and inverter on b. Number the ROM wires $1 \ldots k$, and the non-ROM wires $k+1$ and $k+2$. Let us first suppose that there is at least one uncomplemented literal, and put a $C(1, k+2)$ on the circuit; note that $C(1, k+2)$ applied to the input (x,a,b) gives $(x,a,b \oplus x_1)$. We will write this as $C(1, k+2) : (x,a,b) \rightarrow (x,a,b \oplus x_1)$, and denote this operation by V_1. Then, we define the circuit V_2' as the sequence

Size	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Exor	1	4	6	4	4	12	18	12	6	12	19	16	10	8	10	16	19	12	6	12	18	12	4	4	6	4	1
Opt T	1	4	6	4	4	12	21	24	29	33	44	46	22	5	1	0	0	0	0	0	0	0	0	0	0	0	0
Opt	1	7	21	35	36	28	28	36	35	21	7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2: Circuit size distribution of 3+2 ROM based circuits synthesized using various algorithms.

of gates $T(2, k+2, k+1)V_0 T(2, k+2, k+1)V_0$, and one may check that $V_2' : (x, a, b) \to (x, a \oplus x_1 x_2, b)$. We define V_2 by exchanging the wires $k+1$ and $k+2$; clearly $V_2 : (x, a, b) \to (x, a, b \oplus x_1 x_2)$.

More generally, given a circuit $V_l : (x, a, b \oplus x_1 \ldots x_{l-1}) \to (x, a \oplus x_1 \ldots x_l)$, we define $V_{l+1}' := T(l+1, k+2, k+1)V_l T(l+1, k+2, k+1)V_l$; one may check that $V_{l+1}' : (x, a, b) \to (x, a \oplus x_1 \ldots x_{l+1}, b)$. Define V_{l+1} by exchanging the wires $k+1$ and $k+2$; then clearly $V_{l+1} : (x, a, b) \to (x, a, b \oplus x_1 \ldots x_{l+1})$. By induction, we may get as many uncomplemented literals in this product as we like. □

A careful count of the number of gates used yields the following.

PROPOSITION 11. *If a function's EXOR-SUM decomposition consists of only one term, let k be the number of literals appearing (without complementation) If $k > 0$ then there will be $3 \cdot 2^{k-1} - 2$ gates.*

We applied the above construction [17] to all 256 functions implementable in 2-bit ROM based circuits with 3 bits of ROM. The circuit size distribution is given in the line labeled "Exor" in Table 2. That is compared with optimal circuit sizes produced by the algorithm from Section 4. The line "Opt T" gives the size distribution of circuits synthesized under the restriction [17] that ≤ 1 control bit per gate be on a ROM bit, which is observed by the EXOR-SUM based heuristic. This is why $\forall j$ the sum of the first j numbers in the "Opt T" line is $\geq$ than that in the "Exor" line. Travaglione et al [17] mention that their results do not depend on the above restriction, and the "Opt" line of Table 2 relaxes it.

The "Exor" line of Table 2 took ≈ 2 minutes to generate. Using a circuit library with up to 6 gates (191Mb file, 1.5 min to produce), the "Opt" line took ≈ 5 minutes to generate. Using a 5-gate library improved the runtimes by at least 2x if we do not synthesize the only circuit of size 11. To produce the results in the "Opt T" line, we first found (in 15 min) the 250 optimal circuits of size 12 and less using a 6-gate library (61Mb, 5min). The remaining 6 functions were synthesized in 5 min using a 7-gate library (376Mb, 10 min). This required $>$1Gb or RAM.

Although most functions computable by a 2-bit ROM-based circuit actually require 2 bits [17], there is a simple algorithm for determining if a function may be computed by a 1-bit ROM-based CNT-circuit, and find an optimal circuit if so. It is facilitated by the observation that gates in the circuit may be reordered any way we like, as no gate affects the inputs to the control-bits of any other gate. This means that any given gate will flip the output bit, or not, depending only on the original value of the input bits. Every gate in the CNT library is involutive, so there can be at most one copy of each gate applied to a given subset of wires. Thus, to synthesize the given permutation, we simply check its value on each input combination with 0, 1, or 2 ones in its binary expansion (again, we have relaxed the restriction that only 1 control may be on a ROM wire). If the value of the function is 1, the circuit must have a N, C or T gate controlled by those bits. Note that this gives a way of determining if a given permutation can be synthesized by a 1-bit ROM-based CNT-circuit.

In the case of $k+1$ ROM synthesis, it is clear that adding the S gate to the gate library will never decrease circuit sizes: no two

wires may be swapped since at least one of them is a ROM wire. In the case of $k+2$ ROM synthesis, it is at least intuitively plausible that the same will be true, as if two wires are to be swapped, they have to be the two non-ROM wires – one of which must be returned to its initial value by the end of the computation. We ran an experiment comparing circuit lengths in the 3+2 ROM based case and found no improvement in circuit sizes upon adding the S gate, however we have been unable to prove this in general.

7. CONCLUSIONS

Reversible circuits have numerous applications, from cryptography to subroutines of quantum algorithms. In this work, we study optimal and heuristic synthesis methods of reversible circuits using no temporary storage. We show constructively that all even permutations can be synthesized in this manner, and propose heuristic algorithms for synthesis. We give circuit equivalences which are useful to push NOT gates to one end of the circuit, and possibly for future research on optimization heuristics. We describe an algorithm for the synthesis of optimal circuits, and demonstrate its application to Grover's search in quantum computing.

8. REFERENCES

[1] A. Barenco et al., "Elementary Gates For Quantum Computation", *Physical Review A* **52**, 1995, pp. 3457-3467.

[2] C. Bennett, "Logical Reversibility of Computation", *IBM J. of Research and Development*, **17**, 1973, pp. 525-532.

[3] S. Bettelli, L. Serafini and T. Calarco, "Toward an Architecture for Quantum Programming", Nov. 2001 (version 2), http://arxiv.org/abs/cs.PL/0103009.

[4] L. K. Grover, "A Framework For Fast Quantum Mechanical Algorithms", *ACM Symp. on Theory of Computing (STOC)*, 1998.

[5] T. Hogg et al., "Tools for Quantum Algorithms", http://arxiv.org/abs/quant-ph/9811073, 1998.

[6] P. Kerntopf, "A Comparison of Logical Efficiency of Reversible and Conventional Gates", *IWLS 2000*, pp. 261-269.

[7] R. Korf, "Artificial Intelligence Search Algorithms", *Algorithms and Theory of Computation Handbook*, CRC Press, 1999.

[8] E. Lawler, "An Approach to Multilevel Boolean Minimization", J. of ACM, **11**, No. 3, July 1964, pp. 283-295.

[9] John P. McGregor and Ruby B. Lee, "Architectural Enhancements for Fast Subword Permutations with Repetitions in Cryptographic Applications," *Proc. of ICCD 2001*, Sept. 2001, pp. 453-461.

[10] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge Univ. Press, Sept. 2000.

[11] M. Perkowski et al., "A General Decomposition For Reversible Logic", *Reed-Muller Workshop*, Aug. 2001.

[12] T. Sasao, K. Kinoshita, "Conservative Logic Elements and Their Universality," *IEEE Trans. on Computers*, **28**, '79, pp. 682-685.

[13] T. Silke, "PROBLEM: register swap", December 1995,
`http://www.mathematik.uni-bielefeld.de/`
`~silke/PROBLEMS/bit_swap`

[14] B. Steinbach and A. Mishchenko, "A New Approach to
Exact ESOP Minimization", *Proc. Reed-Muller Workshop*,
August 2001, Starkville, Mississippi, pp. 66-81.

[15] Z. Shi and R. Lee, "Bit Permutation Instructions for
Accelerating Software Cryptography", *IEEE Intl. Conf. on
Application-specific Systems, Architectures, and Processors*,
July 2000, pp. 138-148.

[16] T. Toffoli, "Reversible Computing", *Tech. Memo
MIT/LCS/TM-151*, MIT Lab for Comp. Sci, 1980.

[17] B. Travaglione, M. Nielsen, H. Wiseman, A. Ambainis,
"ROM-based computation: Quantum Versus Classical",
2001.
`http://arxiv.org/abs/quant-ph/0109016`

[18] S. Younis and T. Knight, "Asymptotically Zero Energy
Split-Level Charge Recovery Logic," *Workshop on Low
Power Design*, 1994.

Appendix

Below we state and prove technical results used in Section 3.

PROPOSITION 12. *For $n \geq 5$, we may write any permutation in
A_n as the product of no more than n pairs of disjoint transpositions.*

Proof: Fix $\pi \in A_n$. Then take the cycle decomposition of π and
decompose each cycle into transpositions to write π as a product
of $c(\pi) \leq n$ transpositions. Since π is even, we know $c(\pi) = 2k$
for some k. Pair up the $2i$-th and $(2i+1)$-st transpositions. Some
of these pairs may not be disjoint, but since $n \geq 5$ we may write
$(a,b)(a,c) = (a,b)(d,e)(d,e)(a,c)$ where $d \neq e$ are distinct from
a, b, c. Thus breaking up non-disjoint pairs, we write π as a product
of $2k = c(\pi) \leq n$ pairs of transpositions. $\square$

PROPOSITION 13. *Let $n \geq 4$, and a, b, c, d be distinct integers
between 0 and $n-1$. Then there exists a constructable permutation
$\pi \in A_{2^n}$ such that $\pi(a) = 0$, $\pi(b) = 1$, $\pi(c) = 2$, and $\pi(d) = 3$. It
takes at most $2n$ N gates, $4(n+1)$ C gates, and $2(n-2)$ T gates.*
Proof: Start with an empty circuit and place N gates on every line
corresponding to a 1 in the binary expansion of a. Let π_0 be the
permutation performed by the circuit so far; $\pi_0(a) = 0$.

Since $b \neq a$, so $\pi_0(b) \neq 0$ and therefore $\pi_0(b)$ has at least one
1 in its binary expansion. Say it's on the h-th line; then using C
gates controlled on the h-th line, flip any other non-zero bits of
b'. Finally, if $h \neq 1$, swap the h-th bit and the 0th bit[4]. Let π_1 be
the permutation performed by the circuit so far. by construction,
$\pi_1(b) = 1$, and since C gates fix 0, we have $\pi_1(a) = \pi_0(a) = 0$.

As before, $c \neq b, a \implies \pi_1(c) \neq 1, 0$ hence $\pi_1(c)$ has a 1 some-
where in its binary expansion other than the lowest bit, say in the
k-th bit. Using the algorithm of the previous paragraph, flip every
other bit to 0 and then swap the k-th and 2-nd bit; we note that
again we have not affected 0, and none of our C gates have been
controlled on the bottom line, we cannot move 1. The permutation
π_2 performed by the circuit thus far has the property that $\pi_2(c) = 2$,
$\pi_2(b) = 1$, $\pi_2(a) = 0$.

Finally, observe that $\pi_2(d) \geq 3$; if it is in fact 3 then we are
done, if not then we have $\pi_2(d) \geq 4$, and some bit in the binary
expansion of $\pi_2(d)$ other than the lowest two bits must be 1; let it
be the m-th bit. Then using C gates controlled the m-th bit, flip the

---

[4]This may always be done using 3 C gates. In this case, since we
know that the bottom bit is 0 and the h-th bit is 1, we need only 2.

bottom two wires to 1 if necessary, and use T gates controlled on
these bottom two bits to clear off the rest of the wires. We are now
done, as none of these gates affect $0, 1, 2$, and this subcircuit sends
$\pi_2(d) \to 3$. A careful count of the gates used verifies the final claim
of the proposition. $\square$